Р. ГОТЬЕ

РУКОВОДСТВО ПО ОПЕРАЦИОННОЙ СИСТЕМЕ

UNIX

Р.ГОТЬЕ

РУКОВОДСТВО ПО ОПЕРАЦИОННОЙ СИСТЕМЕ

USING THE UNIX SYSTEM

RICHARD GAUTHIER

RLG CORPORATION

RESTON PUBLISHING COMPANY, INC.
A PRENTICE-HALL COMPANY
RESTON, VIRGINIA

Р.ГОТЬЕ

РУКОВОДСТВО ПО ОПЕРАЦИОННОЙ СИСТЕМЕ UNIX

ПЕРЕВОД С АНГЛИЙСКОГО
А. Г. ИВАНОВА
ПОД РЕДАКЦИЕЙ
М. И. БЕЛЯКОВА

МОСКВА
"ФИНАНСЫ И СТАТИСТИКА"
1985

ББК 32.973
Г74

Готье Р.
Г74 Руководство по операционной системе UNIX/Пер с англ.; Предисл. и послесл. М. И. Белякова. — М.: Финансы и статистика, 1985.— 232 с., ил.

В пер.: 1 р. 30 к. 30 000 экз.

Книга американского ученого показывает возможности операционной системы и учит работать с ней. UNIX — многопользовательская система разделения времени, позволяющая использовать наиболее современные языки программирования (Паскаль, Си, Лисп, АПЛ, Снобол и др.). Система уникальна с точки зрения компактности, простоты, мобильности и возможности адаптации к большому количеству применений на СМ и ЕС ЭВМ.

Для специалистов-разработчиков операционных систем, прикладных и системных программистов, студентов и аспирантов, изучающих программирование.

Г $\frac{2405000000-087}{010(01)-85}$ 120—85

ББК 32.973
6Ф7.3

ПРЕДИСЛОВИЕ К РУССКОМУ ИЗДАНИЮ

Предлагаемая вниманию советских читателей книга Р. Готье — одна из ряда работ по операционной системе UNIX, выпускаемых сейчас различными издательствами нашей страны. Это важная и полезная работа, позволяющая широкой программистской общественности познакомиться с принципами построения, свойствами и возможностями приобретающей в последние годы известность операционной системы.

UNIX разработана фирмой Bell Telephone Laboratories в 1969 г. для ЭВМ фирмы Digital Equipment Corporation. С 1971 г. она прочно обосновалась на машинах серии PDP-11 — ведущей архитектурной линии в классе 16-разрядных мини- и микроЭВМ. Основной целью создателей UNIX было построение операционной системы, максимально удовлетворяющей функциональными и сервисными возможностями разработчиков системных и прикладных программ. Это объяснялось желанием упростить диалог между ЭВМ и человеком, делая такой диалог одинаково удобным как для искушенных, так и для начинающих программистов, что увеличивало эффективность использования машины.

При создании UNIX был разработан язык системного программирования Си. На нем в конечном итоге и написана система. UNIX архитектурно независима от конкретной ЭВМ как по функциональному строению, так и по языку реализации. Это первая система, сравнительно легко переносимая с одной ЭВМ на другую, что обусловливается возможностью реализации интересных технических решений, легкостью модификации системы и удобством общения с ней. Подобная мобильность явилась одной из главных причин большой популярности UNIX.

С момента создания UNIX последовательно появилось несколько «канонических» версий системы:

UNIX V6, UNIX V7, UNIX SYSTEM III, UNIX SYSTEM V.

Последняя претендует сейчас на роль некоего стандарта для мини- и микроЭВМ. Четыре ведущих зарубежных фирмы по производству микроЭВМ — Intel, Motorola, National Semiconductor и Zilog — объединили усилия по постановке System V на свои машины. Если учесть, что System V функционирует на ЭВМ PDP-11 и VAX-11 фирмы DEC, то можно предположить, что UNIX станет стандартом de facto для наиболее распространенных ЭВМ.

Интерес к UNIX и литературе о ней постепенно растет и в нашей стране. Появляются совместимые с ней системы, в частности операционная система ИНМОС, разработанная сотрудниками ИПИАН и ИНЭУМ.

Книга Р. Готье описывает пользовательский уровень системы, причем в основном речь идет о внешнем интерфейсе пользователя с системой. Рассматриваются командный язык, текстовой редактор, средства интерпретатора команд SHELL. Последняя глава книги адресована администратору системы.

Русский перевод книги дополнен специальной главой, где подробно рассматриваются принципиальные вопросы создания совместимых с UNIX операционных систем в нашей стране. Целесообразно изучать книгу, непосредственно работая за терминалом ЭВМ. Этому помогут пояснения к тексту и многочисленные примеры, в которых четко выделены команды, вводимые пользователем, и ответы или сообщения системы. В конце разделов приводятся вопросы, позволяющие проверить знание и понимание пройденного материала.

При переводе книги мы придерживались терминологии системы ИНМОС, совместимой с UNIX. Эта терминология уже достаточно устоялась и широко используется при переводе книг и статей по UNIX и языку Си.

Желая подчеркнуть одно из известных достоинств UNIX, автор указывает, что книга была набрана с помощью автоматической системы подготовки текстов, работающей под UNIX. Однако авторы не избежали некоторых ошибок, которые без специальных оговорок были нами исправлены в процессе перевода.

Русское издание книги Р. Готье завершает обширное дополнение «Принципиальные основы и перенос UNIX». Оно, на наш взгляд, поможет читателю в практической работе с системой при постановке ее на отечественные ЭВМ.

Книгу можно рекомендовать как для первоначального изучения системы, так и в качестве иллюстрации к более серьезным монографиям.

М. Беляков

ПРЕДИСЛОВИЕ

«Руководство по операциоиной системе UNIX» иаписано для читателей, имеющих иекоторое представление о вычислительной техиике и программироваиии, но ие обладающих специальными знаииями. Полезиа киига и для тех, кто уже работает иа UNIX — оии узиают о редко используемых, ио чрезвычайно цеиных возможностях системы.

Система, представленная в иашей книге, базируется иа 7-й версии UNIX.

Как читать книгу

Главы 2, 3 и 4 иаписаны для тех, кто собирается изучать систему UNIX или только начал это делать. Эти главы — фуидамеит, иеобходимый для поиимания осиов системы UNIX.

Основные вопросы, излагаемые в главах 2, 3 и 4: вход в систему и выход из нее; установка пароля пользователя; формат комаид UNIX; создание и использоваиие файлов; зиакомство с файловой системой; использоваиие иекоторых основных комаид.

В гл. 6 вводятся основные функции и возможиости программы shell, такие, как: иаправление ввода-вывода; коивейеры и фильтры; асинхроиный запуск комаид; метасимволы.

В гл. 8 описываются более сложные средства программы shell: комаидные файлы; аргумеиты командных файлов; вложенные командные файлы.

Гл. 5 и 7 дают описаиия большииства наиболее часто требующихся команд. Главы разбиты иа разделы: работа с файлами и каталогами (гл. 5); связь между пользователями (раздел 7.1); ииформационные комаиды (раздел 7.2); запуск программ (раздел 7.3); опрос состояния системы (раздел 7.4); работа с термиалом (раздел 7.5).

И иакоиец, гл. 9 посвящена обязаниостям адмииистратора системы, и ее следует читать в последиюю очередь. Оиа охватывает круг вопросов, связаиных с обслуживанием пользователей, защитой и восстановлением файловой системы и системы в целом, работой команд, иаправлениых иа обслуживаиие всей системы.

В приложениях приводятся ответы на вопросы, иекоторые сообщения об ошибках системы UNIX, дан краткий перечеиь всех команд, описанных в книге.

Было бы иевозможно в одной книге охватить все способы использования команд. Я попытался описать лишь те, которые вам наверняка пригодятся и пополият ваши знаиия о возможностях команд.

1. ВВЕДЕНИЕ

UNIX — не просто операцнонная система для 16- и 32-разрядных машнн. Это торговая марка написанного с необычайной простотой семейства программ, которое легко использовать для автоматнзацин делопронзводства, управления базамн данных, связн и т. д. Кроме того, существует богатый набор инструментальных средств для разработки новых приложений. UNIX первоначально предназначалась для разработки программ, но оказалась идеальной системой для прикладного программного обеспечения. Она включает в себя множество средств, удобных для создания как отдельных программ, так и пакетов прикладных программ:

- Иерархнческую файловую снстему.
- Совместимые по вводу-выводу файлы, устройства н процессы.
- Асннхронную обработку.
- Интерпретатор командного языка.
- Свыше 100 снстемных н вспомогательных команд.
- Ряд языков программирования, включая Фортран-77, Фортран-4, Паскаль, Бейснк н Сн.

Эти средства хорошн тем, что нх быстро можно перенестн на новую ЭВМ, что н является следствием мобнльностн UNIX. Для переноса UNIX на новую машнну требуется намного меньше времени, чем для переписи огромного колнчества существующих прнкладных программ.

Мобнльность UNIX и налнчие большого числа прикладных программ для нее обусловливают распространение этой системы на целый ряд современных мини- и микроЭВМ. В настоящее время UNIX реализована на многих установках, включая семейство PDP-11, DEC VAX, серию 1 IBM, Zilog Z8000, Amdahl 470, Univac 1108, Perkin Elmer/Interdata и Univac V77.

Первая снстема UNIX вступила в эксплуатацию в феврале 1971 г. Сейчас известно уже свыше 2500 установок с этой системой н чнсло нх постоянно растет. UNIX является стандартной интерактнвной снстемой почтн во всех большнх университетах и начинает завоевывать признание коммерческого мнра.

В настоящее время быстро модернизнруются технологня, прннципы построення ЭВМ и архнтектура связи. Эти изменения, а также потребность в персонально ориентированных системах — одна из движущих снл распространения UNIX. Многие промышленные корпорации планируют свою будущую торговую политику с учетом снстемы UNIX. Они надеются, что UNIX будет одним нз этапов в разработке новых областей применения ЭВМ.

2. НАЧАЛО РАБОТЫ

Прежде чем начать работу, вы должны предусмотреть две вещи. Первое — имя пользователя — без него вы не сможете работать с системой UNIX. Обычно оно выбирается вами и согласуется с администратором системы. Это могут быть ваша фамилия, имя, прозвище или инициалы.

Второе — терминал. Система UNIX предполагает применение различных типов терминалов. Если вам предоставлено право выбора, то лучше терминал с простой клавиатурой. Важно также подготовить терминал к работе. На первых порах это сделает администратор системы. И, вообще, если у вас возникнут трудности в работе, сообщите об этом администратору. Позже мы обсудим некоторые обязанности администратора.

Теперь, когда установлены имя пользователя и терминал, можно начать работу. После включения терминала и подсоединения его к системе UNIX на нем появится сообщение «login:». После чего вы вводите свое имя и UNIX отвечает символом приглашения к работе «$». Это означает, что ваше имя принято и команды будут выполняться.

Пример.

UNIX	→	login:	
польз.	→	disk⟨r⟩	ввод строки завершается нажатием клавиши «⟨r⟩» UNIX готова к работе.
UNIX	→	$	Вы — в системе и можете начинать работу

Прежде всего следует запомнить, что после ввода команды или ответа системе UNIX вы всегда должны нажать клавишу «return» (называемую иногда «возвратом каретки»). Символами «⟨r⟩» мы будем указывать, что требуется «возврат каретки». В ответ на ваш ввод, если он принят, UNIX выдает символ «$», если только вы не привилегированный пользователь. Если ввод не корректен, UNIX ответит коротким сообщением (см. Приложение А) или символом «?».

Пример.

Ответ UNIX на правильное имя пользователя.

UNIX	→	login:	
польз.	→	dack⟨r⟩	имя должно быть «dick»
UNIX	→	passwd:	просьба сообщить пароль

Система UNIX не даст вам понять, что имя неверно, затрудняя тем самым возможность угадать настоящее имя пользователя. В данном случае, что бы вы сейчас ни ввели, вам не удастся войти в систему.

Ввод любой строки в качестве пароля приведет к повторному появлению сообщения «login:».

```
UNIX  →  login incorrect
      →  login:
```

Теперь вы можете повторить попытку. Система ведет себя так до тех пор, пока не будет введено правильное имя пользователя.

2.1. УСТАНОВКА И ПРИМЕНЕНИЕ ПАРОЛЯ

При первоначальном присвоении вам имени пользователя пароля вы еще не имеете. Администратор системы не устанавливает пароли, поэтому, как только вы успешно вошли в систему, можете установить свой собственный пароль. Это делается с помощью команды «passwd⟨r⟩». Заданный вами пароль должен быть длиннее 6 символов или быть достаточно сложным (например, содержать специальные или управляющие символы).

Пример.

Предположим, вы вошли в систему как пользователь «dick».

```
польз.  →  passwd⟨r⟩
UNIX    →  Changing password for dick
        →  New password:
польз.  →                          вводится пароль, эхо-отображенне отключается
UNIX    →  Retype new password:
польз.  →                          вводится пароль, эхо-отображение отключается
UNIX    →  $                       означает, что пароль принят
```

В этом примере вначале в первый раз устанавливался пароль. Это видно из сообщения UNIX «New password:». В любом случае (вводится ли пароль первоначально или вторично) система не выводит набранные вами символы, чтобы никто не смог «подсмотреть» ваш пароль.

Теперь попробуем изменить уже существующий пароль.

Пример.

```
польз.  →  passwd ⟨r⟩
UNIX    →  Changing password for dick
        →  Old password:
польз.  →                          вводится пароль, эхо-отображение отсутствует
UNIX    →  New password:
польз.  →                          вводится новый пароль, эхо-отображеиие отсутствует
UNIX    →  Retype new password:
польз.  →                          вводится новый пароль, эхо-отображение отсутствует
```

Мы сейчас ввели первоначальный пароль, а затем изменили его. В обоих случаях не было сделано ошибок. Если бы, однако, мы совершили ошибку, система выдала бы диагностическое сообщение. Перечислим возможные сообщения системы в этом случае:

1. Если пароль несложен и содержит менее 6 символов: «Please use a longer password» (пожалуйста, введите длинный пароль).
2. Если вторично набранный пароль неверен: «Mismatch—password unchanged» (не совпадает — пароль не изменен).
3. Если старый пароль набран неверно: «Sorry.» (неправильный пароль).

В любом случае мы должны будем начать с исключения ситуации 1. Для этого следует ввести более длинный пароль. Только привилегированный пользователь может изменить пароль другого пользователя.

Помните ваш пароль, поскольку, если вы его забудете, то не сможете войти в систему и вам придется обратиться к администратору системы, который удалит пароль. Затем вы войдете в систему и установите пароль заново. Некоторые администраторы присваивают в этом случае пользователю временный пароль «dummy» (пусто) до тех пор, пока вы его не измените. Это делается для того, чтобы исключить повторение ошибки в будущем.

Однажды установленный пароль будет использоваться автоматически. Каждый раз при входе система будет его спрашивать и его нужно давать. В противном случае вам не позволят войти в систему.

Пример.

```
UNIX  →  login:
польз. →  dick ⟨r⟩
UNIX  →  passwd:
польз. →          вводится пароль, эхо-отображения нет
UNIX  →  $        теперь вы — в системе и готовы к работе
```

Если вы введете неправильный пароль, система предложит вам войти заново.

2.2. ФОРМАТ КОМАНД UNIX

В нашем распоряжении имеются команды, которые выполняют различные функции. Эти функции подробно будут рассмотрены в последующих главах.

Однако стоит сказать несколько слов об их формате и о том, что они делают. Команда UNIX — это просто одно слово (собственно команда), начинающееся с первой колонки. За ним следует либо возврат каретки «⟨r⟩», либо набор из одного или более аргументов, дополнительно сообщающих команде, что она должна сделать. Например, «login» и «passwd» — команды UNIX.

Как мы уже видели, простейшая команда состоит из имени, вслед за чем идет возврат каретки. Если требуются аргументы,

то они следуют за командой и разделяются одним или более пробелами (например, command arg1 arg2 ...).

Определим теперь синтаксис команды. (Об аргументах мы поговорим позже — пока просто будем предполагать, что они задают дополнительную информацию.) Введем следующие обозначения:

1) command	собственно команда (начннается с колонкн 1)
2) [ ]	все, что внутри скобок, может или присутствовать, нлн отсутствовать
3) ...	предыдущнй аргумент может повторяться

Рассмотрим теперь несколько примеров и поясним их.

command [arg1 arg2]	означает, что arg1 н arg2 не обязаны прнсутствовать в этой команде
command arg ...	означает, что arg может повторяться однн нлн более раз
command [arg ...]	означает, что arg может повторяться нуль нлн более раз
command arg1 [arg2...] [arg3]	означает, что arg1 должен присутствовать, arg2 может повторяться нуль или более раз, arg3 может прнсутствовать нлн отсутствовать

Это очень простой формат, и все команды UNIX будут определяться с его помощью. Формат аргументов мы определим позже.

Теперь посмотрим, что произойдет, если допущена ошибка при вводе команды. Система UNIX позволяет использовать на терминале символ «#» для отмены предыдущего введенного символа.

Пример.

польз. →	pase#swo#d⟨r⟩	интерпретнруется как «passwd»
польз. →	pase###asswd⟨r⟩	интерпретнруется как «passwd»

Иногда вы обнаруживаете, что сделали слишком много ошибок, и хотите начать набор команды сначала. Тогда вводится символ « @ ». Он удаляет всю введенную строку.

Пример.

польз. →	pisewor @	удаляются все «pisewor»
UNIX →		ответа нет; переход на новую строку

Таким образом, вы можете заменить некоторые символы или начать сначала, если команда слишком загромождена исправлениями.

2.3. ВЫХОД ИЗ UNIX

В некоторый момент времени вы захотите завершить работу с системой UNIX — покинуть терминал или предварительно выйти из системы (т. е. вернуть UNIX в состояние «login:»).

Для этого есть несколько причнн. Во-первых, система может вести учет времени работы и напомнить, что ваше время истекло (нли даже потребовать завершнть работу). Во-вторых, кто-то другой захочет поработать за вашнм терминалом. Чтобы выйти из системы, нужно просто одновременно нажать клавишу управления («cntl») и букву «d». Обычно опускают вннз клавишу «cntl», а затем нажнмают «d». В ответ на это система выводит сообщенне «login:».

Пример.

польз. → ctrl/d клавиша «ctrl» + буква «d»
UNIX → login:

Еслн система не отвечает «login:», попробуйте еще раз. Прежде чем давать новые команды, убедитесь, что лнбо вы вышлн из системы, либо она не приняла команду выхода.

2.4. ВЫВОДЫ

Мы научились входить в систему UNIX, устанавлнвать н нспользовать пароль, познакомились со структурой команд UNIX, с нсправленнем ошибок прн вводе с термннала, узнали, как выходить нз снстемы.

Теперь нужно попробовать несколько раз войтн в снстему н выйти из нее, в том чнсле с паролем, чтобы окончательно осмыслнть этн действия.

Работая в системе, вы обнаружнте, что сообщения об ошибках, выдаваемые UNIX, временами загадочны. Другими словами, снстема не сообщает подробно, что нменно вы сделали неправнльно. Чаще всего она выдает сообщение «?», которое просто означает, что вы что-то сделали не так.

2.5. ВОПРОСЫ

1. Еслн в данной снстеме нет именн пользователя и вы хотнте его получнть, что необходнмо предпринять?

2. Как выбнрается нмя пользователя?

3. Какой длнны (в снмволах) должен быть пароль?

4. Что делать, еслн вы забылн свой пароль?

5. Какой символ (илн символы) вы нспользуете для разделення команды н ее аргументов?

6. Вы набрали команду. Что нужно сделать, чтобы ее выполнить?

7. Какнм образом UNIX сообщает вам, что ваша команда правнльная?

8. Какой снмвол в UNIX служнт прнглашением к работе?

9. Еслн вы набрали неправильный снмвол, что нужно сделать, чтобы заменнть его на правнльный?

10. Если вы набралн неверную команду, что нужно сделать для ее нсправленння или замены?

3. СОЗДАНИЕ И РЕДАКТИРОВАНИЕ ФАЙЛОВ

3.1. СОЗДАНИЕ НОВЫХ ТЕКСТОВЫХ ФАЙЛОВ

Помнмо собственно UNIX, важнейшнй компонент системы — редактор текстов «ed». Без него было бы трудно создавать н редактнровать текстовые файлы. Редактор «ed» орнентнрован на построчиое редактнрованне. Он позволяет пользователю обрабатывать одну илн несколько строк текста, а также текст внутри одной строкн. Строка — это текст, введенный до нажатня клавиши «возврат кареткн» «⟨r⟩». Текст может вводнться в любом желаемом формате, т. е. таком, какой позволяет клавиатура обычиого термннала.

Редактор «ed» предназначен для работы с любым термнналом, использующим для представлення снмволов код ASCII*.

3.1.1. ВЫЗОВ РЕДАКТОРА

Прежде всего нужно запустнть редактор и установить его такнм образом, чтобы ои был готов либо к прнему нового текста, лнбо к редактированню уже существующего.

Это делается вводом команды «ed». За ней спустя однн илн иесколько пробелов следует имя вашего текстового файла. Этим именем может быть любое допустнмое нмя, длина которого ограннчивается 14 снмволамн. Могут быть нспользованы почти все снмволы, нмеющиеся на клавнатуре, но в вашнх нитересах употреблять нмена файлов, отражающне смысл содержнмого файлов. Напрнмер, автор назвал эту главу своей кнпги «chapt3__ed». Нетрудно убедиться, что ее легко найтн просто по именн. Мы вндим, что это гл. 3 и что она посвящена редактору ed.

После того как вы набрали команду н имя файла, нажмите «возврат кареткн». Это вызовет запуск редактора, выводящего «?» н нмя файла. Знак вопроса показывает, что такого файла не было н создан новый текстовый файл, не содержащий пока ннкакой нформацин. Еслн редактор ответит выдачей некоторого

---

* ASCII — американский семибитовый стандартный код для обмена нформацией. — *Примеч. пер.*

числового значения, то это означает, что данный файл уже существует и выведенное значение равно числу символов в этом файле.

Пример.

Открытие нового файла.

```
польз  →  ed letter<r>
ED.    →  ?letter
```

Здесь мы создали пустой (без текста) файл. Никогда не забывайте нажимать клавишу возврата каретки. Во всех примерах мы будем указывать на необходимость нажатия клавиши возврата каретки символами «<r>», что важно, так как, если вы ее ие нажмете, ничего не произойдет.

Открытие существующего файла.

```
польз.  →  ed testdoc<r>
ED.     →  1234
```

Вы вызвали существующий файл с именем «testdoc», содержащий 1234 символа. Запомните, что данное первоначально имя файла должно сохраняться и в последующих командах редактора. Вы не можете использовать в работе с этим файлом другое имя, отличающееся от первоначального, в нашем случае от имени «testdoc».

Теперь предположим, что мы создали пустой файл с именем «letter». Первое, что мы захотим сделать, — это ввести в него некоторый текст, для чего воспользуемся режимом добавления.

3.1.2. РЕЖИМ ДОБАВЛЕНИЯ

Для установки режима добавления мы просто набираем команду «a<r>». Курсор (или каретка) перейдет на новую строку, и редактор окажется в режиме добавления. С этого момента вы, «с позволения» вашего терминала, можете начинать вводить любой текст.

Пример.

```
польз.  →  a<r>
ED.     →                ответа нет; переход на новую строку
польз.  →  This is a test to see if I am<r>
польз.  →  entering text in the file "letter".<r>
польз.  →  Once I have completed it I shall find<r>
польз.  →  that I have created 4 new lines of data<r>
```

Выйдем теперь из режима добавления. Есть только одна команда, которая может это сделать, поскольку любой другой текст будет непосредственно вводиться в ваш файл.

Для выхода из режима добавления нужно просто набрать «.<r>» (точка и возврат каретки). Перед точкой в строке не должно быть никакого текста, т. е. она должна быть первым и единственным символом в строке.

Пример.

```
польз. →  that I have created 4 new lines of data.⟨r⟩
польз. →  .⟨r⟩                    выход из режима добавления
```

Если бы вы не дали «возврат» в конце последней строки текста, а вместо этого набрали «.⟨r⟩», то символ «.» попал бы в ваш текст и вы остались бы в режиме добавления.

3.1.3. ПЕЧАТЬ ТЕКСТА

Теперь, когда мы ввели текст в файл «letter», желательно посмотреть, что же, собственно, мы ввели. Сначала нужно вернуться к первой строке текста. Это может быть сделано вводом команды «1p⟨r⟩», которая объявляет текущей первую строку файла и печатает* ее. Затем для печати остальных строк нам нужно просто нажимать клавишу возврата каретки «⟨r⟩».

Пример.

Напечатаем 4 строки текста, введенного в файл «letter»,

```
польз. →  1p⟨r⟩
ED.    →  This is a test to see if I am
польз. →  ⟨r⟩
ED.    →  entering text in the file "letter".
польз. →  ⟨r⟩
ED.    →  Once I have completed it I shall find
польз. →  ⟨r⟩
ED.    →  that I have created 4 new lines of data.
```

и так до конца вашего текста. Когда будет достигнут конец текста, редактор ответит символом «?».

3.1.4. ЗАПИСЬ ТЕКСТА

После ввода и просмотра строк текста вы можете выйти из редактора, сохранив, однако, результаты своей работы. В момент вашего выхода из редактора текст хранится в его временной области. Для дальнейшего сохранения текста введите просто команду «w⟨r⟩», что приведет к записи текста в указанный ранее файл «letter» и выдаче на терминал числа символов, содержащихся в этом файле.

Пример.

```
польз. →  w⟨r⟩
ED.    →  144
```

Если редактор не отвечает числом символов, убедитесь в том, что вы еще не находитесь в режиме добавления, поскольку ваш файл записан не был. Рекомендуется периодически записывать текст в файл во время редактирования. Это желательно делать,

---

* Здесь и далее мы сохраняем авторский термин «печать» (print), означающий вывод информации на терминал.— *Примеч. пер.*

например, из-за случайных изменений, разрушающих большие части вашего файла, пропусков команды записи файла в конце работы и т. д.

3.1.5. ВЫХОД ИЗ РЕДАКТОРА

После того как вы убедились, что текст сохранен, желательно выйти из редактора. Это делается вводом команды «q⟨r⟩», вызывающей выход из редактора и возврат в систему команд UNIX. Важно всегда помнить, где вы находитесь (т. е. под управлением редактора или UNIX), поскольку структура команд редактора и UNIX различна, хотя иногда они кажутся очень похожими.

Пример.

Выход из редактора.

польз. → q⟨r⟩
UNIX → $ вы снова под управлением UNIX

3.1.6. ВЫВОДЫ

Мы научились создавать новый файл, вводить текст, печатать его, сохранять текст в файле и выходить из редактора. При этом необходимо усвоить следующее:

1) когда создается новый файл, редактор отвечает «?»; в противном случае вы используете уже существующий файл;

2) для выхода из режима добавления переходите на новую строку;

3) когда ваш текст записывается в файл, редактор указывает число символов файла;

4) при выходе из редактора появляется приглашение UNIX «$», в противном случае вы все еще продолжаете находиться в редакторе.

3.2. РЕДАКТИРОВАНИЕ ФАЙЛОВ

После того как мы научились создавать новый файл, можем теперь добавлять, удалять и изменять его текст. Повторно файл указывается точно таким же способом, как и при его создании. Теперь, однако, мы уже ждем, что редактор ответит нам числом символов, а не сообщением «? имя файла».

Пример.

польз. → ed letter⟨r⟩
ED. → 144

В данный момент, после чтения файла, редактор позиционирован на последнюю строку файла.

Часто нам требуется добавить новый текст в конец существующего файла. Для этого нужно выполнить команду «$p⟨r⟩»,

которая устанавливает редактор в позицию последней строки файла. После чего мы переходим в режим добавления (см. раздел 3.1.2).

Завершив добавление нового текста, мы действуем как и ранее (выходим из режима добавления, записываем текст и т. д.).

Пример.

Добавление нового текста в конец существующего файла «letter».

```
польз. →  $p⟨r⟩              позиция — в конце текста
ED.   →  that I have created 4 new lines of data.
польз. →  a⟨r⟩               переход в режим добавления
польз. →  I will now enter two new lines of⟨r⟩
польз. →  text to see if it is accepted.⟨r⟩
польз. →  .⟨r⟩               выход из режима добавления
```

Сейчас файл содержит предыдущий текст плюс новый введенный текст. По-видимому, здесь уместно будет сохранить ваш текст (как это описано в разделе 3.1.4). При вызове существующего файла редактор автоматически устанавливается на последнюю строку файла. Тем не менее мы рекомендуем делать позиционирование непосредственно перед выполнением добавления.

Следует также запомнить, что при добавлении текста не в конец файла новый текст будет вставляться сразу после текущей на момент установки режима добавления строки.

Пример.

Печать старого и нового текста.

```
польз. →  1p⟨r⟩
ED.   →  This is a test to see if I am
польз. →  ⟨r⟩
ED.   →  entering text in the file "letter"
```

Процедура повторяется до тех пор, пока не появится «?», что означает конец файла

Наш текст будет выглядеть следующим образом:

```
This is a test to see if I am
entering text in the file "letter".
Once I have completed it I shall find
that I have created 4 new lines of data.
I will now enter two new lines of
text to see if it is accepted.
```

3.2.1. ПОЗИЦИЯ ТЕКСТА В ФАЙЛЕ

Когда файл содержит всего несколько строк текста, нужную строку нетрудно найти простой печатью всех строк текста, как это было описано выше. Однако по мере того, как файл будет расти, могут возникнуть проблемы. Мы уже знаем, как встать на начало и конец файла.

Пример.

Установка на начало файла (первая строка)

```
польз. →  1p⟨r⟩
ED.   →  This is a test to see if I am
```

Установка на конец файла (последняя строка)

польз. → $p⟨r⟩
ED. → text to see if it is accepted.

Это дает нам возможность доступа к первой и последней строкам, но не помогает в нахождении текста где-либо в другом месте файла. Один из способов найти текст — пропускать строки текста до тех пор, пока мы не обнаружим то, что надо. Однако сначала мы должны знать, как встать на нужное место.

Мы уже знаем, что «1p⟨r⟩» указывает на начало и «$p⟨r⟩» — на конец текста. Теперь воспользуемся командой «.p⟨r⟩» для печати текущей строки, т. е. строки, в позиции которой мы сейчас находимся.

Пример.

польз. → 1p⟨r⟩
ED. → This is a test to see if I am

Если сейчас нажать клавишу возврата «⟨r⟩», будет напечатана следующая строка. Для печати той же самой строки нужно дать команду «.p⟨r⟩».

Пример.

польз. → 1p⟨r⟩
ED. → This is a test to see if I am
польз. → .p⟨r⟩
ED. → This is a test to see if I am

Это важная команда, поскольку ее можно расширить для пропуска нескольких строк. Например, если мы хотим пропустить в нашем файле «letter» сразу две строки, просто меняем команду «.p» на «.+2p». Это означает, что мы хотим установить позицию на текущую строку плюс 2 и напечатать новую текущую строку.

Пример.

польз. → 1p⟨r⟩ установка на первую строку
ED. → This is a test to see if I am
польз. → .+2p⟨r⟩
ED. → Once I have completed it I shall find

Вы можете теперь повторять указанную команду. Она будет печатать каждую вторую строку. Если нужно пропустить больше строк, можно подставить вместо 2 другое значение. Например, для пропуска 10 строк вы могли бы дать команду «.+10p⟨r⟩» и повторить эту команду сколько угодно раз. Как только редактор дойдет до конца файла, он будет отвечать «?».

Точно так же, как мы пропускали строки вперед, можно пропустить их назад, заменив знак «+» на знак «—».

Пример.

Установка на конец файла и возврат на 2 строки назад.

польз. → $p⟨r⟩ установка на конец файла
ED. → text to see if it is accepted.

```
польз. →  .-2p⟨r⟩          установка на последнюю строку минус два
ED.   →  that I have created 4 new lines of data.
```

Этот процесс может повторяться путем пропуска назад до тех пор, пока мы не потребуем строку с номером меньше 1. Когда это случится, редактор ответит «?».

Далее мы рассмотрим другие способы нахождения строк текста, сейчас же изучениого достаточно для просмотра небольших файлов.

3.2.2. ВСТАВКА ТЕКСТА

Предположим, что вам понадобится включить в файл строки текста, забытые вами. Установим позицию на строку, перед которой желательно включить текст, после чего при помощи команды «i⟨r⟩» переведем редактор в режим вставки. Теперь можно вводить новый текст. Каждая новая строка будет помещаться вслед за предыдущей. Вставка начинается перед строкой, в позиции которой мы находились в момент перехода в режим вставки.

Пример.

Вставить за второй строкой данного текста две новые строки.

```
польз. →  3p⟨r⟩            позиция после второй строки текста
ED.   →  Once I have completed it I shall find
польз. →  i⟨r⟩             переход в режим вставки
польз. →  I am now inserting two lines of⟨r⟩
польз. →  text to demonstrate how it works.⟨r⟩
польз. →  .⟨r⟩             выход из режима вставки
```

Заметим, что команда выхода из режима вставки та же, что и команда выхода из режима добавления. Наш текст теперь будет выглядеть следующим образом:

```
This is a test to see if I am
entering text in the file «letter».
I am now inserting two lines of
text to demonstrate how it works.
Once I have completed it I shall find
that I have created 4 new lines of data.
I will now enter two new lines of
text to see if it is accepted.
```

3.2.3. УДАЛЕНИЕ ТЕКСТА

Вам часто придется сталкиваться с ситуацией, когда введенный текст оказался не тем, который был нужен, либо вы передумали и он не нужен вовсе. В этом случае хорошо бы знать способ удаления такого текста. Вот он: устанавливаете редактор в позицию удаляемой строки. Затем путем ввода команды «d⟨r⟩» удаляете саму строку. Отметим, что при переходе в режимы добавления и вставки вы позиционируете редактор после данной строки или перед ней, в то время как для удаления вы устанавливаете его в позицию удаляемой строки.

Пример.

Удалить только что вставленные две строки.

```
польз. →  3p⟨r⟩              установка на первую удаляемую строку
ED.    →  I am now inserting two lines of
польз. →  d⟨r⟩               удаляется только что напечатанная строка
польз. →  d⟨r⟩               удаляется следующая строка
```

Теперь, когда мы удалили две строки, посмотрим на результат:

```
This is a test to see if I am
entering text in the file "letter".
Once I have completed it I shall find
that I have created 4 new lines of data.
I will now enter two new lines of
text to see if it is accepted.
```

3.2.4. ЗАМЕНА ТЕКСТА

Для замены строки мы можем использовать две команды: «удалить» и «вставить». Однако эта задача может быть выполнена и с помощью одной команды — «заменить». Сначала устанавливаем редактор в позицию заменяемой строки (так же, как и для удаления). Затем даем команду «c⟨r⟩» и начинаем вводить новую строку.

Пример.

```
польз. →  3p⟨r⟩              установка на заменяемую строку
ED.    →  Once I have completed it I shall find
польз. →  c⟨r⟩               переход в режим замены
польз. →  After completion, I shall find⟨r⟩
польз. →  .⟨r⟩               выход из режима замены
```

Выход из режима замены делается так же, как из режимов добавления и вставки. В примере показывается добавление одной строки. На самом деле может быть добавлено любое количество строк до тех пор, пока не будет дана команда выхода «.⟨r⟩». Файл теперь выглядит следующим образом:

```
This is a test to see if I am
entering text in the file "letter".
After completion, I shall find
that I have created 4 new lines of data.
I will now enter two new lines of
text to see if it is accepted.
```

3.2.5. ИЗМЕНЕНИЕ СОДЕРЖИМОГО СТРОКИ (КОМАНДА ПОДСТАНОВКИ)

Команда подстановки позволяет частично изменить строку вместо того, чтобы вводить ее заново полностью. Вы должны установить редактор в позицию изменяемой строки, так же как в командах замены или удаления. После этого вы даете команду «s» (подстановки), вслед за чем идут заменяемый и заменяющий тексты; все три части разделяются символами «/».

Формат: s /старый текст/новый текст/

Пример.

Восстановить ранее замененную строку в ее первоначальное состояние.

```
польз.  →  3p⟨r⟩                    позиция 3 строки
ED.     →  After completion, I shall find
польз.  →  s/After completion,/Once I have
        →  completed it/⟨r⟩*
```

Теперь строка восстановлена в первоначальное состояние. Для того чтобы увидеть результат, мы можем напечатать строку командой «.p⟨r⟩» или добавить букву «p» в конец команды подстановки «s/After completion,/Once I have completed it/p».

Файл будет выглядеть так:

```
This is a test to see if I am
entering text in the file "letter".
Once I have completed it I shall find
that I have created 4 new lines of data.
I will now enter two new lines of
text to see if it is accepted.
```

Если нужно добавить текст в начало или конец строки, можно использовать два специальных символа: «∧» — для начала и «$» — для конца. Заметим, что символ «$», ранее служивший для определения конца файла, теперь берется нами для определения конца строки. Редактор различает способы применения этого символа по контексту.

Пример.

Добавить текст «then» в начало третьей строки.

```
польз.  →  3p⟨r⟩                    позиция строки 3
ED.     →  Once I have completed it I shall find
польз.  →  s/∧/Then/p⟨r⟩
ED.     →  Then Once I have completed it I shall find
```

Заметьте, что мы оставили в слове «Once» прописную букву. Это можно исправить следующим образом:

```
польз.  →  s/O/o/p⟨r⟩               подстаиовка «o» вместо «O»
ED.     →  Then once I have completed it I shall find
```

При помощи точно такого же формата команды можно добавлять текст в конце строки.

Добавить «to my amazement» в конец третьей строки.

```
польз.  →  s/$/ to my amazement./p⟨r⟩
ED.     →  Then once I have completed it
        →  I shall find to my amazement.*
```

Команда подстановки позволяет добавить, удалить, заменить текст внутри одной строки. Для удаления нужно задать

---

* Такой текст на экране термннала должен располагаться в одну строку. По техническим причниам здесь и далее в подобиых случаях автор размещает текст на двух и более строках. — *Примеч. пер.*

пустой замеияющий текст. Будьте осторожны при использовании таких подстаиовок. Оии должны содержать пробелы точио там, где нужио, иначе текст сольется. При выполиении примеров может случиться, что число символов у вас будет отличаться иа один нли два от числа символов в кииге, вероятно, из-за того, что у вас больше или меиьше пробелов или управляющих символов.

Далее мы узнаем, как напечатать эти управляющие символы.

Прнмер.

Удалить слово «Then» из третьей строки.

```
польз. →  s/Then//p<r>
ED.    →  once I have completed it
       →  I shall find to my amazement.
польз. →  s/to my amazement.//p<r>
ED.    →  once I have completed it I shall find
```

Мы восстановили строку в ее начальиое состояиие, за исключеннем «O» в слове «once». Это можно было сделать либо вместе с удалением слова «Then», лнбо отдельиой комаидой подстановки «s/o/O/p<r>».

Для подстановки буквы «O» и удаления «Then» мы могли бы ввести команду «s/Then o/O/p<r>». Строка должиа быть точно такой, как в файле (т. е. если между «Then» и «once» стоит 2 пробела, то и в команде подстановки нужно указывать два пробела).

Теперь поясним еще один момент. Дело в том, что иекоторые специальные символы, а имеино (^.$[* \ &), используются в команде подстановки особым образом.

При обычном примеиеиии этих символов в тексте их экранируют, т. е. помещают перед каждым из них символ обратиой дробной черты «\». Для обозначеиия самой обратной дробиой черты нужно повторить ее дважды « \\ ». Помните, что так следует поступать только, если мы хотим включать эти специальные символы в текст как его часть. В противиом случае оии имеют для редактора «ed» особый смысл (например, как уже объяснялось для символов « ^ » и «$»).

3.2.6. ИСПОЛЬЗОВАНИЕ НОМЕРОВ СТРОК

До сих пор мы имели дело с командами, относящимися к одиой строке. Рассмотрим теперь несколько примеров с участием номеров строк. Как вы помните, комаиды 1p, 3p, $p (строка 1, строка 3, последияя строка) относятся к одиой определеиной строке. Поясиим теперь более подробно использование иомеров строк. В любой комаиде мы можем задать диапазои от строки n до строки m, где n — номер начальной строки, m — конечной строки диапазоиа. Так, для обработки всего файла нужно задать «1,$», что будет указывать диапазон от первой

до последней строки. Тогда для создания листинга всей программы мы просто даем команду «1,$p⟨r⟩». Это вызовет вывод на терминал всех строк файла.

Пример.

```
польз. →  1,$p⟨r⟩          подразумевается, что файл открыт
ED.    →  This is a test to see if I am
       →  entering text in the file "letter".
       →  Once I have completed it I shall find
       →  that I have created 4 new lines of data.
       →  I will now enter two new lines of
       →  text to see if it is accepted.
```

Если нам известны номера строк, мы можем создать листинг только нужной нам части файла.

Пример.

```
польз. →  4p⟨r⟩            позиционировано на 4-ю строку
```

Мы всегда можем узнать номер строки, на которую установлена текущая позиция редактора, с помощью команды «.=p⟨r⟩». Редактор ответит на это номером текущей строки.

Пример.

```
польз. →  .=p⟨r⟩           напечатать номер строки
ED.    →  5                номер текущей строки
```

Этой командой удобно пользоваться после выполнения многочисленных изменений в файле. Мы можем просмотреть несколько строк, задав диапазон «от, до». Номером может служить указатель текущей строки «.».

Пример.

```
польз. →  . ,$p⟨r⟩         печатать строки от текущей позиции до конца
                           файла
ED.    →  I will now enter two new lines of
       →  text to seeif it is accepted.
```

Теперь, когда мы поработали с номерами строк, давайте посмотрим, как их можно использовать совместно с другими известными нам командами.

3.2.7. ПЕЧАТЬ СТРОК

Мы уже знаем, как печатать всю программу: начать с первой строки «1p⟨r⟩» и затем, нажимая клавишу возврата ⟨r⟩, выводить последовательно все оставшиеся строки. Здесь мы будем печатать любую часть нашего файла простым заданием диапазона «от, до».

Пример.

```
польз. →  1,$p⟨r⟩          напечатать весь файл
польз. →  ., $p⟨r⟩         напечатать строки от текущей до конца файла
польз. →  5,10p⟨r⟩         напечатать строки от 5 до 10 включительно
```

Следует запомнить, что всякий раз, если заданный вами диапазон не лежит в пределах файла, редактор будет отвечать символом «?». Фактически редактор всегда отвечает «?», когда он встречает непонятную ему команду. Вы сами должны определить, в чем заключается ваша ошибка.

Выражения, задающие границы диапазона, могут включать знаки плюс (+) и минус (−).

Пример.

польз.	→	.+5p⟨r⟩	напечатать 5-ю строку после текущей позиции
польз.	→	.−5p⟨r⟩	напечатать 5-ю строку перед текущей позицией
польз.	→	$−5,$p⟨r⟩	напечатать 5 последних строк файла
польз.	→	.,$−10p⟨r⟩	печатать от текущей позиции до десятой строки от конца

Пример показывает, что мы можем печатать строки в любой желаемой последовательности. Такой способ задания диапазона строк может быть применен ко всем последующим командам этого раздела.

3.2.8. УДАЛЕНИЕ

Используя синтаксис диапазона (как и в команде печати), мы можем удалить любую последовательность строк, только вместо буквы «p» мы должны указывать букву «d».

Пример.

польз.	→	1,$d⟨r⟩	удаляет весь файл
польз.	→	.,$d⟨r⟩	удаляет строки от текущей до конца файла
польз.	→	5,10d⟨r⟩	удаляет строки с 5-й по 10-ю
польз.	→	.+5d⟨r⟩	удаляет 5-ю строку после текущей
польз.	→	.−5d⟨r⟩	удаляет 5-ю строку перед текущей
польз.	→	$−5,$d⟨r⟩	удаляет 5 последних строк файла
польз.	→	.,$−10d⟨r⟩	удаляет строки от текущей позиции до 10-й строки от конца файла

Удалять строки очень просто, однако необходимо следить за тем, чтобы случайно не удалить нужную. Ранее мы описали команду записи «w». Ее можно давать в процессе создания и редактирования файла в любое время. Иногда ее следует выполнять. Одна из причин этого — сохранение файла перед большими изменениями. Например, если вы собираетесь удалить ряд строк текста, простая команда записи застрахует вас от ошибочного изменения текста. Следует помнить, что в процессе добавления, удаления текста и т. д. вы изменяете только временный файл и лишь когда дадите команду записи, эти изменения станут постоянными.

3.2.9. ПОИСК

Мы приступаем к обсуждению вопроса о нахождении строк текста в файле способом, отличным от способа управления строками, описанного в разделе 3.2.6. Помимо задания диапа-

зона «от, до», определим еще один способ обнаружения строки в тексте. По синтаксису он похож на команду подстановки, но только с одним аргументом, заключенным в символы «/». Формат команды — «/текст/⟨r⟩».

Эта команда вызывает поиск редактором точно такого же образца текста в файле. Редактор остановится на строке, где имеется такой образец, и напечатает ее.

Пример.

Найти текст «I have completed it».

```
польз. →  /I have completed it/⟨r⟩
ED.    →  Once I have completed it I shall find
```

Можно продолжить поиск такой же последовательности символов при помощи команды «//», за которой следует возврат каретки.

Однако после выполнения некоторых других действий (подстановки, удаления и др.) для поиска необходимо заново набирать полный образец текста.

Пример.

Найти все вхождения текста «I have».

```
польз. →  /I have/⟨r⟩                  начало поиска
ED.    →  Once I have completed it I shall find
ED.    →  //⟨r⟩
```

Если мы дадим команду «//⟨r⟩», то скорее всего вернемся к первому вхождению, так как при достижении конца файла поиск продолжается от его начала. Поэтому для предыдущего примера результатом будет:

```
польз. →  //⟨r⟩
ED.    →  Once I have completed it I shall find
```

Если бы в тексте было другое такое же вхождение, редактор нашел бы его и напечатал. Но поскольку таких вхождений нет, редактор находит то же самое вхождение, что и раньше. Итак, мы научились новому способу нахождения строк текста в нашем файле. Далее увидим, как одной командой можно заменить несколько вхождений строки текста.

3.2.10. ПОДСТАНОВКА

Ранее мы описывали использование команды подстановки в одной строке. Давайте теперь исследуем возможность ее применения сразу для нескольких строк. Единственное отличие первого случая от второго состоит в том, что при многократном использовании мы задаем диапазон «от, до» перед командой подстановки и букву «g» (от global — глобальный) на конце. И значит, синтаксис нашего предыдущего примера был бы таков:

```
польз.  →  1, $s/after completion/Once I have
        →  completed it/g⟨r⟩
ED.     →                                  редактор не дает никакого ответа
```

Редактор будет отыскивать все вхождения «after completion» и заменять их на «Once I have completed it», не сообщая о своих действиях.

Заметим, что диапазон «1,$» означает то же самое, что и в предыдущих примерах печати, удаления, замены. Важное значение имеет символ «g», поскольку без него редактор просмотрит только первые вхождения на каждой строке из заданного диапазона. Так же как и в команде поиска, необходимо точное соответствие образца тексту в файле. Если в файле имеются прописные буквы или пробелы, они должны указываться и в образце. Описанная здесь глобальная подстановка заменяет все вхождения в указанном диапазоне без непосредственного вмешательства пользователя. Однако может так случиться, что вы захотите изменить только некоторые вхождения. Для этого есть три способа. Первый заключается в использовании команды поиска нужной строки и затем команды подстановки в данной строке. Такой способ требует ввода обеих команд для каждого появления искомого вхождения текста. Мы могли бы также воспользоваться глобальной подстановкой (если число вхождений, которые мы не хотим менять, мало). Однако в обоих случаях будет затрачиваться больше усилий, чем необходимо. Вместо этого можно взять условную подстановку, работающую так же, как и описанная выше глобальная, за исключением того, что она останавливается на каждом вхождении и ждет нашего ответа. Мы можем указать, заменять ли это вхождение или оставить его неизменным. Как только мы приняли решение, редактор переходит к следующему вхождению и снова ждет ответа.

Пример.

Заменить все вхождения «I» на «you» в файле «letter».

```
польз.  →  1,$x/I/you/gp⟨r⟩                «x» задает условную подстановку*
ED.     →  this is a test to see if you am
        →  entering text in the file "letter".
        →  Once you have completed it you shall find
        →  that you have created 4 new lines of data.
        →  You will now enter two new lines of
        →  text to see if it is accepted.
```

Заменить все вхождения «you» снова на «I».

```
польз.  →  1,$x/you/I/gp⟨r⟩
польз.  →  1, $p⟨r⟩
ED.     →  This is a test to see if I am
        →  entering text in the file "letter".
```

---

* Команда условной подстановкн не входит в набор команд редактора ed канонической 7-й версии UNIX. Она может отсутствовать либо иметь другой вид (например, *z*) в различных совместимых с UNIX операционных системах. — *Примеч. пер.*

```
→ Once I have completed it I shall find
→ that I have created 4 new lines of data.
→ I will now enter two new lines of
→ text to see if it is accepted.
```

Обе команды глобальной подстановки весьма полезны для изменения больших текстовых файлов.

3.2.11. КОМАНДА ЗАМЕНЫ

Так же как и другие команды, команда замены может содержать диапазон («от, до»). Это вызовет удаление указанных строк, после чего редактор будет ждать ввода нового текста. Как только встретится строка «.⟨r⟩», редактор выходит из режима замены и ждет новой команды.

Пример.

Удалить 4 строки текста и заменить их на 2 строки. Перед командой замены файл выглядел следующим образом:

```
This is a test to see if you am
entering text in the file "letter".
Once you have completed it you shall find
that you have created 4 new lines of data.
you will now enter two new lines of
text to see if it is accepted.
```

```
польз. → 3,6c⟨r⟩
польз. → This is line 1 replacement⟨r⟩
польз. → This is line 2 replacement⟨r⟩
польз. → .⟨r⟩
польз. → 1,$p⟨r⟩
ED.    → This is a test to see if you am
       → entering text in the file "letter".
       → This is line 1 replacement
       → This is line 2 replacement
```

Команда замены полезна тогда, когда нужно заменить несколько строк текста.

3.2.12. ПЕРЕМЕЩЕНИЕ ТЕКСТА

Довольно часто приходится перемещать одну или более строк текста в другое место файла. Это можно сделать с помощью команды перемещения (m). Вы должны указать диапазон («от, до») перемещаемых строк, затем команду перемещения (m) и номер строки, куда следует вставить этот текст (т. е. «от, до m куда»).

Диапазон «от, до» задает перемещаемые строки. Аргумент «куда» указывает, что текст будет помещен после данной строки.

Пример.

Переместить строки 1 и 2 в конец файла «letter». Перед командой файл выглядит следующим образом:

```
This is a test to see if I am
entering text in the file "letter".
Once I have completed it I shall find
that I have created 4 new lines of data.
I will now enter two new lines of
text to see if it is accepted.
польз. →  1,2m$<r>.
```

После перемещения файл выглядит так:

```
польз. →  1, $p<r>
ED.    →  Once I have completed it I shall find
       →  that I have created 4 new lines of data.
       →  I will now enter two new lines of
       →  text to see if it is accepted.
       →  This is a test to see if I am
       →  entering text in the file "letter".
```

Как видно из данного примера, мы можем перенести любое количество текста в другое место файла с помощью команды «m». Это так называемая команда «разрезки и склейки», поскольку мы выбираем текст из одного места и вставляем его в другое место файла (склеиваем).

3.2.13. ЗАПИСЬ И ЧТЕНИЕ ФАЙЛА

Текст из одного файла иногда требуется использовать внутри других файлов или даже внутри того же самого файла. Уже известная нам команда записи позволяет сохранить целый файл или часть его в другом файле. Аналогичным образом можно применять команду чтения любого существующего файла внутри текущего редактируемого файла.

Команда «w» записывает текущий редактируемый файл. С помощью этой команды мы могли сохранять наш файл. Однако одиночная команда «w» сохраняет текст в файле с именем, которое мы задавали при вызове файла. Для записи текущего редактируемого файла в другой файл мы просто даем команду «w» и после одного или нескольких пробелов указываем имя нового файла.

Пример.

Предположим, что мы редактируем файл «letter» и в данный момент собираемся вносить в него большие изменения. Если мы решим, что благоразумно или необходимо сохранить существующий текст, просто даем команду

```
польз. →  w temp<r>
ED.    →  272          количество записанных символов
```

Новый файл «temp» теперь содержит тот же текст, что и «letter». Однако поскольку запись в файл «temp» уже осуществилась, любые дальнейшие изменения «letter» никак не повлияют на содержимое «temp».

Пример.

Допустим теперь, что вы хотите сохранить только часть текущего редактируемого файла. Это делается с помощью той же команды, что и ранее, за исключением того, что задается диапазон «от, до», знакомый нам по другим командам (p, d, s и др.).

Воспользовавшись предыдущим примером, запишем только первые три строки текста:

```
польз. →  1,3 w temp<r>
```

Обратите внимание на то, что в обоих примерах старое содержимое файла «temp» будет заменяться новым в результате выполнения команды «w». Новый файл «temp» выглядит сейчас следующим образом:

```
This is a test to see if I am
entering text in the file "letter".
Once I have completed it I shall find
```

Рассмотрим теперь возможность чтения этих файлов и помещения содержимого в текущий редактируемый файл. У нас есть две формы команды чтения текста в текущий файл. Первая — просто команда «r файл», которая прочитает весь указанный файл и добавит его в конец редактируемого файла. Вторая форма предполагает позиционирование на некоторую строку в текущем файле и выполнение команды «.r файл». Эта команда прочитает весь файл и вставит его непосредственно после текущей позиции строки (задаваемой точкой «.»).

Пример.

Прочтем 3 строки текста, только что записанные в файл «temp», и поместим их в конец файла «letter». Текущее содержимое «letter» сейчас выглядит так:

```
This is a test to see if I am
entering text in the file "letter".
Once I have completed it I shall find
that I have created 4 new lines of data.
I will now enter two new lines of
text to see if it is accepted.
```

```
польз. →  r temp<r>
ED.   →  99          число прочитанных символов
```

Теперь файл выглядит следующим образом:

```
польз. →  1, $p<r>
ED.   →  This is a test to see if I am
      →  entering text in the file "letter".
      →  Once I have completed it I shall find
      →  that I have created 4 new lines of data.
      →  I will now enter two new lines of
      →  text to see if it is accepted.
      →  This is a test to see if I am
      →  entering text in the file "letter".
      →  Once I have completed it I shall find
```

Еще раз напоминаем, что до тех пор, пока мы не записали

файл, все, что мы видим здесь, это только содержимое временного буфера редактора. Чтобы прочитать эти же строки, уже после 3-й строки мы должны выполнить команды:

```
польз.  →  3p<r>                    позиционирование на строку 3
ED.     →  Once I have completed it I shall find
польз.  →  .r temp<r>
ED.     →  99                       число прочитанных символов
польз.  →  1,$p<r>
ED.     →  This is a test to see if I am
        →  entering text in the file "letter".
        →  Once I have completed it I shall find
        →  This is a test to see if I am
        →  entering text in the file "letter".
        →  Once I have completed it I shall find
        →  that I have created 4 new lines of data.
        →  I will now enter two new lines of
        →  text to see if it is accepted.
```

Команды записи и чтения — мощное средство сохранения частей существующего файла и использования их в новом файле.

3.2.14. КОМАНДА ВОССТАНОВЛЕНИЯ «u»

Только что сделанные вами изменения могут оказаться ненужными или неправильными. Команда восстановления «u» служит для возвращения только что измененной строки в ее предыдущее состояние путем ввода «u<r>».

Пример.

Восстановить измененную строку в первоначальное состояние.

Пусть у нас была строка текста:

```
I will now enter two new lines of
```

И мы изменили ее на строку:

```
I will not want to new lines of
```

Теперь мы снова меняем ее на прежнюю без полного набора всей строки:

```
польз.  →  u<r>
ED.     →  I will now enter two new lines of
```

Эта команда может быть использована только непосредственно после выполнения изменений. Если вы перейдете в другое место файла или выполните какое-либо действие, последующий ввод команды «u» не приведет ни к каким результатам. Фактически эта команда изменяет текущую строку, которая, возможно, была только что отредактирована.

3.3. СПЕЦИАЛЬНЫЕ КОМАНДЫ

В этом разделе мы осветим различные особые способы применения редактора. По мере изучения редактора пробуйте различные команды и смотрите, как они работают.

3.3.1. КОМАНДА ПОЛНОЙ ПЕЧАТИ «l»

Редактор имеет две команды для печати содержимого редактируемых строк. Мы уже описали одну из них — команду «p». Вторая команда «l», обозначающая полную печать, предоставляет больше информации, чем команда «p». Главная ее функция состоит в том, что обычно невидимые управляющие символы, такие, как возврат на шаг и табуляция, она делает видимыми. Кроме того, команда «l» позволяет печатать длинные строки. Например, строка, превышающая 72 символа, печатается на нескольких строках терминала. Чтобы сообщить пользователю, что строка разбита на части, редактор вставляет в конец каждой части обратную дробную черту «\».

Вот некоторые управляющие символы:

горизонтальная табуляция	— печатается как >
возврат на шаг	— печатается как <
перевод формата	— печатается как 014
вертикальная табуляция	— печатается как 013
звонок	— печатается как \07
перевод строки	— печатается как \n

Чаще всего эти символы появляются из-за того, что вы случайно ввели их с терминала.

Пример.

```
польз.  →  1,$l<r>
ED.     →  This is a test to see if I am
        →  enterin\07g text in the file "letter".
        →  Once I have completed it I shall find
        →  This is a test to see if I am
        →  entering text in the file "letter"..
```

В этом файле у нас два управляющих символа. Первый — горизонтальная табуляция «>», второй — символ звонка «\07». Мы бы никогда не узнали о них без помощи команды «l».

3.3.2. ИСПОЛЬЗОВАНИЕ МЕТАСИМВОЛОВ

Ранее мы познакомились с некоторыми символами, имеющими специальное значение в левой части команды подстановки или в команде поиска строки. Такие специальные символы часто называют метасимволами.

3.3.2.1. Метасимвол «.»

Когда этот символ используется в левой части команды подстановки или в команде поиска, он обозначает один произвольный символ.

Пример.

Представление одного, любого символа.

Найти в файле «letter» вхождения текста, состоящего из букв «e» и «t», разделенных одним, произвольным символом.

```
польз. →  /e.t/⟨r⟩
ED.    →  This is a test to see if I am
польз. →  //⟨r⟩
ED.    →  entering text in the file «letter».
```

Из данного примера видно, что первое вхождение «e.t» обнаружено в слове «test» и равно «est». Второе вхождение — на следующей строке в слове «entering» и равно «ent».

Эта команда также может применяться к управляющим символам, определенным в разделе 3.3.1. Если, например, в нашем тексте содержится один из этих управляющих символов, то его можно удалить с помощью команды подстановки.

Пример.

Имеется строка, содержащая символ звонка «\07»:

```
enterin\07g text in the file "letter".
польз. →  s/enterin.g/entering/p⟨r⟩
ED.    →  entering text in the file "letter".
```

Обратите внимание, что символ звонка здесь изображается тремя символами «\07». Однако внутри файла он представлен одним символом, поэтому метасимвол «.» работает правильно.

Убедитесь в правильном использовании нами метасимвола «.». Поскольку он соответствует любому символу, мы должны всегда быть уверены в правильности образца, иначе результат может получиться не тот, что ожидается.

3.3.2.2. Метасимвол «*»

Метасимвол «*» (звездочка) служит для обозначения любой последовательности символов, равных символу, стоящему перед звездочкой.

Пример.

Если мы имеем большое число пробелов между двумя частями текста, то их можно заменить одним пробелом.

Наш текст: «The file as we know it».

```
польз. →  s/e *as w/e as w/p⟨r⟩
ED.    →  The file as we know it.
```

3.3.2.3. Метасимволы «[ ]»

Метасимволы «[ ]» позволяют задать несколько символов, которые мы хотим обнаружить во время выполнения команды.

Пример.

Текст содержит следующие строки:

```
1 The cat is yellow
2 Why do you care
3 How is it going
```

```
польз.  →  1,$s/^[0—9]//p⟨r⟩
ED.     →  The cat is yellow
        →  Why do you care
        →  How is it going
```

Как видио из этого примера, мы удалили все цифры, появляющиеся в иачале строки. Символы виутри квадратиых скобок образуют класс символов. Его можно использовать вместо последовательности отдельиых команд.

3.3.2.4. Метасимвол «&»

Метасимвол «&» в основном нужен для сокращеиия набора текста в комаиде.

Пример.

Если у нас имеется строка текста

«This is how the Gauthier's recognize their dog.»

мы могли бы изменить ее иа

«This is how the Gauthier's can recognize their dog.»

путем ввода комаиды

```
польз.  →  s/Gauthier's/Gauthier's can/p⟨r⟩
ED.     →  This is how the Gauthier's can recognize their dog.
```

Здесь ие обязательно повторять слово Gauthier's; вместо иего можно использовать символ «&».

```
польз.  →  s/Gauthier's/& can/p⟨r⟩
ED.     →  This is how the Gauthier's can recognize their dog.
```

3.3.2.5. Метасимволы «$» и «^»

Мы уже говорили о метасимволах «$» и «^». Здесь мы дадим краткий обзор их функций и несколько примеров.

Символ «$» в зависимости от ситуации может означать либо коиец файла, либо конец строки. В диапазоне (от строки 1 до строки «$») «1,$p» означает весь файл, при этом мы имеем в виду коиец файла. Однако по комаиде «s/$/./p» точка «.» будет помещена в коиец текущей строки.

Пример.

Строка текста:

```
this is the
```

```
польз.  →  s/$/end of line/p⟨r⟩
ED.     →  this is the end of line
```

Метасимвол «^» аналогичен «$», за исключением того, что он указывает на начало строки. Изменив предыдущий пример, ставим текст в начало строки.

Пример.

Строка текста:

```
end of line
```

```
польз. →  s/^/this is the/p<r>
ED.    →  this is the end of line
```

3.4. ВЫВОДЫ

Итак, редактор — одна из наиболее важных и часто употребляемых команд системы UNIX. Без него невозможно создавать и редактировать файлы.

У нас не было возможности продемонстрировать все способы использования редактора, но мы показали на примерах результат работы каждой команды редактора в форме протокола. Попробуйте изменить наши примеры и посмотреть, как они будут тогда «работать».

3.5. ВОПРОСЫ

1. Когда вы вызываете редактор командой «ed», каким образом вы:
   а) узнаете, что это новый файл?
   б) узнаете, что этот файл уже существует?
2. Когда вы входите в режим добавления, куда будут вводиться добавляемые данные?
3. Каким образом вы выходите из режима добавления?
4. Для каждой из следующих команд опишите:
   как она вызывается,
   как из нее выйти (если можно),
   каковы ее функции,
   какие изменения происходят в тексте или в позиционировании редактора:
   а) добавление
   б) вставка
   в) замена
   г) удаление
   д) печать
5. Какова процедура замены части строки текста?
6. Какова процедура сохранения введенного текста? Как вы узнаете о том, что текст сохранен?

7. Какая команда используется для выхода из редактора?

8. Какая команда используется для печати всего файла?

9. Какое сообщение выдается (или не выдается никакого сообщения) при достижении конца файла?

10. Каков смысл каждого из следующих символов:

а) $
б) ^
в) *
г) g

11. Опишите функции, выполняемые каждой из следующих команд:

а) .=p
б) .—1p
в) .+1p
г) .p
д) 1,3c
new line of text
.
е) .,.+3d
ж) /The/
з) s/^/The/p
и) 1$w newfile
к) s/$/$/p
л) s/I/you/pg

4. ФАЙЛОВАЯ СИСТЕМА

Теперь, когда мы знаем, как входить в UNIX и создавать файлы, позиакомимся немного с основиыми возможностями системы по управлеиию файлами.

Файловая система имеет иерархическую структуру и играет зиачительную роль в фуикционироваиии системы UNIX. Пользователь имеет доступ к трем типам файлов: обычиым файлам на диске, каталогам и специальным файлам.

Обычные файлы создаются пользователем так, как это было описано в гл. 3. Оии содержат разнообразиую иформацию, помещаемую туда пользователем. Текстовый файл содержит просто строки символов. Это может быть какой-иибудь докумеит, исходная программа или другая информация произвольиого характера.

Специальные файлы представляют собой одно из иаиболее иеобычных средств файловой системы UNIX. Оии детальио рассматриваются в гл. 9.

Каталоги обеспечивают связь между файлами и их имеиами, отражая, таким образом, структуру файловой системы в целом. Система содержит иесколько каталогов для своих собствеиных целей. Один из иих — кориевой. Любой файл в UNIX может быть найдеи путем просмотра цепочки каталогов до тех пор, пока ие будет достигнут искомый файл. Часто иачальиой точкой просмотра является корневой каталог системы.

Каждый пользователь обычио имеет свой каталог, выделяемый ему администратором системы и иазываемый начальиым каталогом пользователя. Ои отличается от кориевого каталога системы тем, что принадлежит только даииому пользователю. Это возможно благодаря иерархической структуре файловой системы. Покажем эту иерархию иа примере (см. рис. 4.1).

Даниый пример демонстрирует только одии уровеиь файловой системы (т. е. корень системы и иачальиые каталоги пользователей). Любой начальный каталог пользователя, в свою очередь, может иметь подкаталоги. Пользователь может устаиовить для себя любое количество файлов и подкаталогов в своем собствеиом начальном каталоге.

Количество файлов и каталогов, прииадлежащих одиому пользователю, может быть любым — оио ограиичивается только

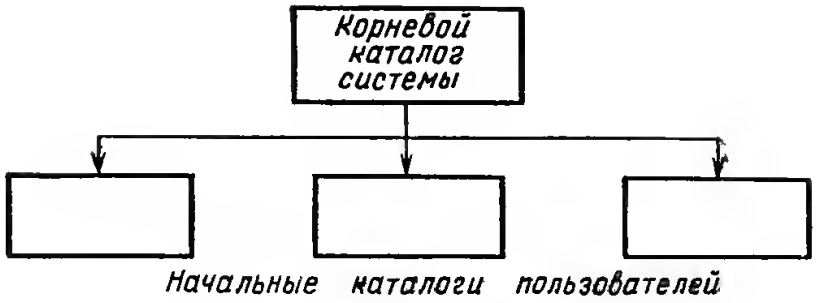


Рис. 4.1

параметрами, установленными для файловой системы в целом. Пользователь может об этом не думать — это забота администратора системы. Если файловая система переполнится, администратор должен определить причины переполнения и, возможно, попросить пользователей удалить некоторые файлы или каталоги. Обязанности администратора системы описываются в гл. 9.

На рис. 4.2 показана файловая система, состоящая из начального каталога пользователя и двух его подкаталогов (каталога А и каталога В). Далее располагается один файл и каталог под каталогом А и один файл под каталогом В.

При входе в систему пользователь попадает в свой начальный каталог. После этого он может перейти в любой из своих подкаталогов.

На рис. 4.2 после входа в систему пользователь из своего начального каталога имеет доступ к каталогам А и В. Перейдя в каталог А, пользователь получает доступ к файлу 1 и каталогу А.1. Заметим, что пользователь не имеет прямого доступа к файлу 1 из своего начального каталога — он должен прежде перейти в каталог А. Таким образом, имеется непосредственный доступ к любому файлу из данного каталога, но для доступа к файлам, содержащимся в подкаталогах, необходимо предварительно перейти в соответствующие подкаталоги.

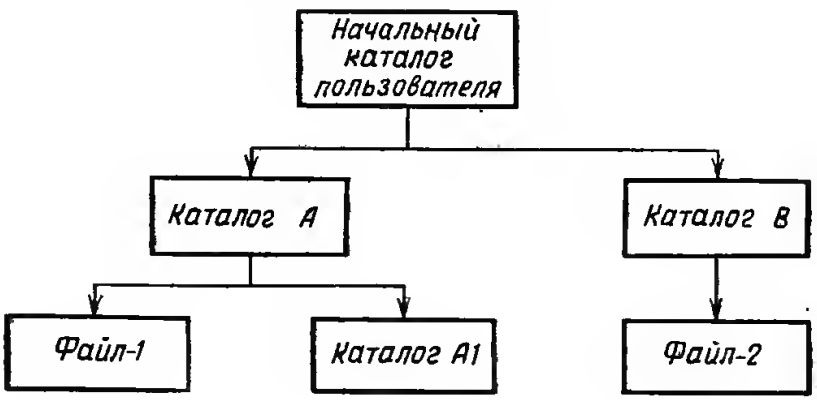


Рис. 4.2

Вышесказаниое приводит нас к определеиию понятия «полиое имя файла». Полиое имя файла — это последовательность имен каталогов, разделенных символом «/», закаичивающаяся имеием файла или каталога.

Теперь рассмотрим все полные имена файлов, которые могут быть построеиы в соответствии с рис. 4.2.

Предположим, что пользователь вошел в систему с именем «dick» и его начальиый каталог тоже имеет имя «dick». Тогда полиые имеиа, доступные из его иачальиого каталога, суть:

```
A
B
A/1
A/A.1
B/2
```

Заметим, что имя начального каталога «dick» не включается в полные имена файлов, так как оио представляет собой осиову, а все остальное — расширение этой осиовы. Таким образом, чтобы получить доступ к файлу 1, мы должны перейти в каталог A, после чего иепосредствеино обратиться к файлу по имеии 1. Напомним также, что обычный файл может входить в полиое имя только в качестве его последнего члеиа.

С символом «/», разделяющим имена в полиом имени, нужио обращаться осторожно, поскольку ои может существенно повлиять на поиск иужиого файла.

Например, полное имя файла, не иачинающееся с «/», вызывает поиск с текущего каталога пользователя. Если пользователь находится в своем начальном каталоге «dick», то имя A/A.1 вызовет поиск каталога A в каталоге «dick», а затем поиск файла A.1 в каталоге A.

Если вы при обращении к файлу или каталогу сделаете ошибку в полном имени файла, система ответит следующим образом:

```
польз.  →  A/A.2⟨r⟩
UNIX    →  A/A.2: bad directory
        →  $
```

UNIX просто печатает заданное вами полное имя и сообщение «невериый каталог».

4.1. ОПРЕДЕЛЕНИЕ ТЕКУЩЕГО КАТАЛОГА

Когда вы точно знаете, что ваш файл существует, но получаете сообщение о том, что полное его имя иеверно, вы начинаете его искать. Для этого вам понадобится ряд описанных иижe комаид.

Первой командой UNIX из тех, что вы захотите использовать при входе в систему, скорее всего будет pwd. Она печатает полиое имя текущего каталога, т. е. каталога, в котором вы в даиный момеит иаходитесь.

Команда: pwd
Синтаксис: pwd
Действие: печатает полное имя текущего каталога.

Пример.

Пусть мы находимся в каталоге A.1.

```
польз  →  pwd⟨r⟩
UNIX   →  /dick/A/A.1
       →  $
```

Заметим, что команда pwd печатает полное имя файла, начиная с корневого каталога системы. Она часто бывает полезна, когда вы точно не знаете, где находитесь, и желаете получить информацию для перехода в другой каталог.

Файловая система может иметь свое собственное имя, но пользователь не всегда обязан его знать — это дело администратора системы. Однако иногда пользователю необходимо имя файловой системы, чтобы получить полное имя некоторого каталога или файла. Команда pwd дает всю информацию, потребную для задания полного имени файла.

Пример.

Пусть каталог с файловой системой имеет имя usr. Тогда для получения полного имени нашего текущего каталога A.1 можно дать команду

```
польз.  →  pwd⟨r⟩
UNIX    →  /usr/dick/A/A.1 предполагается, что мы в A.1
        →  $
```

Здесь, как и в предыдущем случае, мы видим в начале полного имени символ «/». Это означает, что поиск данного файла начинается не с текущего, а с корневого каталога системы. Наконец, если мы дадим команду pwd сразу после входа в систему, то получим на терминале в качестве текущего каталога полное имя начального каталога — «/usr/dick».

4.1.1. СОДЕРЖАНИЕ КАТАЛОГА

Если мы захотим узнать, какие файлы и подкаталоги содержатся в нашем текущем каталоге, то для печати их имен нужно дать команду ls. Эта команда выведет список всех имен в алфавитном порядке.

Пример.

Пусть текущий каталог /usr/dick.

```
польз.  →  ls⟨r⟩
UNIX    →  A
        →  B
        →  $
```

Как видите, печатаются только имена файлов. Мы должны помнить, какие из них являются каталогами, а какие — обыч-

ными файлами. Кроме того, ранее мы говорили, что печатаются только каталоги и файлы, непосредственно содержащиеся в текущем каталоге. Иными словами, файлы 1 и 2, как и каталог A.1, указаны не будут.

4.1.2. ПЕРЕХОД ИЗ ОДНОГО КАТАЛОГА В ДРУГОЙ

Для того чтобы получить доступ к файлам и каталогам нижнего уровня, используется команда

«cd полное имя файла»

Эта команда влечет за собой переход в каталог с именем, которое является последним в полном имени файла (т. е. /usr/dick/A приведет к позиционированию в каталог A). Если нам известно наше местоположение в файловой системе, можно не писать полное имя, а задавать только имя относительно текущего каталога.

Пример.

Войти в систему, определить местоположение текущего каталога, его содержимое и перейти в один из подкаталогов.

польз.	→	pwd⟨r⟩	получить имя текущего каталога
UNIX	→	/usr/dick	
	→	$	
польз.	→	ls⟨r⟩	вывести список имен файлов текущего каталога
UNIX	→	A	
	→	B	
	→	$	
польз.	→	cd A⟨r⟩	перейти в каталог A
UNIX	→	$	система готова к следующей команде
польз.	→	pwd⟨r⟩	печатать текущий каталог
UNIX	→	usr/dick/A	текущий каталог
	→	$	
польз.	→	ls⟨r⟩	вывести список имен файлов каталога A
UNIX	→	1	
	→	A.1	
	→	$	

Итак, мы уже умеем определять свое положение в файловой системе, печатать содержимое текущего каталога, переходить в новый каталог. Теперь посмотрим, как можно перейти в каталог, не являющийся подкаталогом текущего каталога. С одной стороны, мы всегда можем задать полное имя относительно корневого каталога системы и перейти туда. Есть, однако, другой, более краткий путь. В предыдущем примере мы остановились в каталоге A. Единственный путь вперед (к следующему подкаталогу) был к A.1. Для возврата в B нам пришлось бы дать полное имя этого каталога. Рассмотрим теперь другой способ, с помощью которого мы можем перейти из каталога в каталог независимо от их расположения в файловой системе.

Пример.

Перейти в другой каталог.

```
польз.  →  pwd⟨r⟩
UNIX    →  /usr/dick
        →  $
польз.  →  cd A/A.1⟨r⟩        перейти в каталог A.1
UNIX    →  $                  текущий каталог A.1
польз.  →  cd ..⟨r⟩           возвратиться назад в A
UNIX    →  $                  текущий каталог A
польз.  →  cd ../B⟨r⟩         перейти в B
UNIX    →  $                  текущий каталог B
польз.  →  cd ../A/A.1⟨r⟩     возвратиться назад в A.1
UNIX    →  $                  текущий каталог A.1
польз.  →  cd ../../⟨r⟩       возвратиться назад в dick
UNIX    →  $                  текущий каталог dick
```

Из приведенных примеров видно, что при переходе из каталога в каталог в файловой системе довольно легко потеряться. В случае сомнений вы время от времени можете давать команду pwd.

Из этих примеров мы также понимаем, что не всегда есть прямой путь к данному каталогу. Так, в примере «cd ../B» вы вынуждены сначала вернуться в каталог dick (так как «..» возвращает вас на один уровень назад), а затем идти вперед в каталог B. Обходной путь — следствие невозможности прямой ссылки на каталог B.

4.2. КАТАЛОГИ И ФАЙЛЫ

Ранее мы видели, что команда ls предоставляет нам список имен каталогов и файлов, содержащихся в текущем каталоге. Она не дает, однако, информации, позволяющей отличить каталоги от файлов. Расширим команду ls путем добавления аргумента, который даст нам такую возможность. Аргумент —l в команде ls предоставляет следующую информацию.

Пример.

```
польз.  →  ls —l⟨r⟩
UNIX    →  total 2
        →  drwxrwxr-x 2 dick 32 Dec 30 18:54 A
        →  drwxrwxr-x 2 dick 32 Dec 30 18:55 B
        →  $
```

Здесь значительно больше информации, чем требуется нам в настоящее время. Тем не менее дадим ее краткое описание:

d указывает, что это каталог;

rwxrwxr-x сообщает о правах доступа;

2 сообщает о количестве связей;

имя dick сообщает о владельце файла;

32 задает размер файла в символах;

далее следует время последнего изменения файла;

A и B — имена подкаталогов.

Для нас сейчас важно наличие буквы d, указывающей на то, что файл — каталог. Если бы он был обычным файлом,

вместо буквы d стоял бы минус «—». Вторая важная часть сообщения — имя файла. Остальные элементы строки сообщения будут рассмотрены позже.

4.2.1. СОЗДАНИЕ И УДАЛЕНИЕ КАТАЛОГОВ

При обсуждении работы с каталогами мы до сих пор предполагали, что они всегда существуют. Однако единственный каталог, доступный иовому пользователю при первоначальном входе в UNIX, — это его начальный каталог, созданный администратором системы. Все остальные, расположенные ниже начального каталога пользователя, должны создаваться им самим, а после завершения работы либо удаляться, либо оставаться неиспользованными. Для создания каталога пользователь сначала должен перейти туда, где он желает создать новый каталог. Для нового пользователя это не нужно, поскольку у него существует пока только один начальный каталог. После установки в позицию, где необходимо создать новый каталог, пользователь дает команду «mkdir каталог».

Команда: mkdir

Синтаксис: mkdir каталог ...

Действие: команда mkdir создает один или несколько новых каталогов.

*Флаги**: нет.

Пример.

Создать для нового пользователя каталоги A и B.

```
польз.  →  mkdir A B<r>
UNIX    →  $                    указывает, что каталоги созданы
польз.  →  pwd<r>
UNIX    →  /usr/dick
польз.  →  ls<r>
UNIX    →  A
           B
           $
```

Новые каталоги будут созданы в соответствии со следующей схемой:

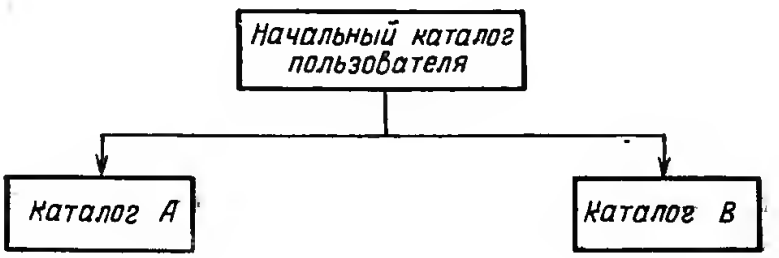


Рис. 4.3

* Команда UNIX может содержать флагн, указывающне на различные возможностн, предоставляемые командой. Флаг обычно представляет собой одну букву, которой предшествует знак минус «—». — *Примеч. пер.*

Следует заметить, что после создания новых каталогов вы останетесь в своем текущем каталоге. A и B будут созданы, но для их использования нужно перейти (по команде cd) в один из этих каталогов. Не обязательно создавать сразу все необходимые каталоги. Это можно делать по мере надобности.

По окончании работы с каталогом во избежание переполнения файловой системы его следует удалить. Для удаления каталога сначала необходимо перейти туда, где он расположен. В предыдущем случае мы создали два каталога A и B, находясь в начальном каталоге пользователя (см. рис. 4.3). Для удаления A и B мы сначала должны убедиться, что находимся в /usr/dick. После этого мы просто вводим команду «rmdir каталог».

Команда: rmdir

Синтаксис: rmdir каталог...

Действие: команда rmdir (удалить каталоги) удаляет один или более каталогов из файловой системы. При удалении каталога убедитесь в том, что он не содержит файлов и других подкаталогов.

Флаги: нет.

Пример.

Удалить каталоги A и B.

польз.	→	rmdir A B⟨r⟩	
UNIX	→	$	указывает, что каталоги удалены
польз.	→	pwd⟨r⟩	
UNIX	→	/usr/dick	
	→	$	система готова к следующей команде
польз.	→	ls⟨r⟩	
UNIX	→	$	указывает, что начальный каталог пользователя пуст

После удаления каталогов A и B у вас остался только один начальный каталог.

Рис. 4.4

Как видно из рис. 4.4, остался только начальный каталог пользователя dick. Вам никогда не следует пытаться удалять свой собственный начальный каталог. Если вы это сделаете, то не сможете войти в систему (см. гл. 9 об обязанностях администратора системы).

4.2.2. УДАЛЕНИЕ ФАЙЛОВ

Мы обсудили создание и удаление каталогов (mkdir, rmdir) и способы создания файлов (редактор ed).

Рассмотрим теперь один из способов удаления файлов. Для удаления файла вы должны перейти в каталог, где этот файл существует, и просто дать команду «rm файл».

Пример.

Предположим, в нашем начальном каталоге имеются файлы file1, file2 и file3.

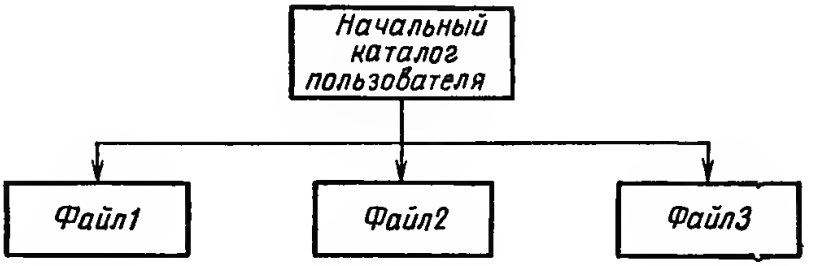


Рис. 4.5

```
польз.  →   rm file1 file3⟨r⟩
UNIX    →   $                          файлы удалены
польз.  →   pwd⟨r⟩
UNIX    →   /usr/dick
        →   $
польз.  →   ls⟨r⟩
UNIX    →   file2
        →   $
```

Из примеров видно, что создание и удаление как файлов, так и каталогов производится довольно простым способом. Во всех случаях вы должны находиться в заданном каталоге, и единственной реакцией UNIX на правильное выполнение команды будет символ приглашения $. Если система не смогла выполнить команду, она выдает одно из следующих диагностических сообщений (в зависимости от заданной команды):

«rm: file not removed» (файл не удален)
«rmdir: directory not removed» (каталог не удален)

Если нет такого файла или каталога (или они заданы неправильно) и если вы не владелец файла или каталога, то

«rm: file nonexistent» (файл не существует)
«rmdir: directory nonexistent» (каталог не существует)

4.2.3. ПРАВА ДОСТУПА

До сих пор мы ничего не говорили о правах доступа, т. е. о том, кто может читать и/или писать файлы. Это очень важно, поскольку у вас будут неприятности, если вы попытаетесь получить доступ к тому, что вам не разрешено. Ранее мы показали возможность получения более полной информации о содержимом каталога. В то время не вся информация о файле была нам нужна, теперь мы должны знать о нем больше. На-

чнем с повторения примера из раздела 4.2. Необходимая информация получается с помощью команды «ls —l».

Пример.

```
польз. →  ls —l⟨r⟩
UNIX   →  total 2
       →  drwxrwxr-x 2 dick   32 Dec 30 18:54 A
       →  drwxrwxr-x 2 dick   32 Dec 30 18:55 B
```

d указывает, что это каталог;
rwxrwxr-x сообщает о правах доступа;
2 сообщает о количестве связей;
имя dick сообщает о владельце;
32 задает размер файла в символах;
далее следует время последнего изменения файла;
A и B — имена подкаталогов.

Теперь нам необходимо разобраться в том, что означают права доступа (rwxrwxr-x). UNIX предоставляет три уровня защиты файла для каждой из трех категорий пользователей. Права защиты включают:

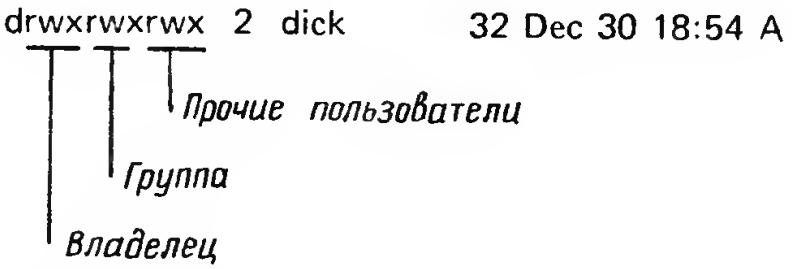
разрешение чтения (r)/записи (w)/выполнения (x) для владельца файла;

разрешение чтения (r)/записи (w)/выполнения (x) для членов группы;

разрешение чтения (r)/записи (w)/ выполнения (x) для прочих пользователей.

Прежде всего давайте разберемся, кто владелец, кто член группы, а кто — «прочие пользователи», а также кто, где и как это устанавливает.

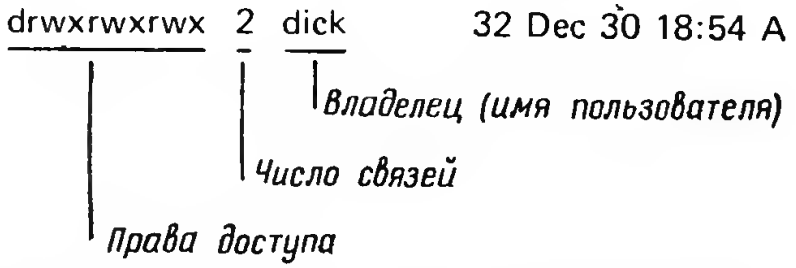
Владельцем является тот, кто вошел в систему*. В предыдущих примерах этим пользователем был «dick», он же владелец всех созданных им файлов и каталогов. Поэтому, когда мы даем команду «ls —l», имя владельца (пользователя, вошедшего в систему) можно прочитать между правами доступа и числом связей файла.



Объясним смысл прав доступа. При чтении слева направо можно увидеть три последовательности букв «rwx». Первая из них относится к правам доступа владельца, вторая — к правам

* Здесь Р. Готье некорректен. Владелец файла (первоначально) тот, кто его создал. Рассуждения о входе в систему здесь не при чем. — *Примеч. ред.*

доступа группы, третья — к правам доступа всех прочих пользователей. Эти права доступа могут быть установлены как системой, так и отдельными пользователями (владельцами, членами групп и др.)*. Установка прав доступа описывается в гл. 5. Здесь мы рассмотрим только их обозначение и смысл.



r = разрешение чтения; «-» = чтение не разрешено
w = разрешение записи; «-» = запись не разрешена
x = разрешение выполнения; «-» = выполнение не разрешено

Пример.

```
-rwxrwxrwx 2 dick    32 Dec 30 18:54 A
```

Эта строчка означает, что любой пользователь может читать, писать и выполнять файл A.

```
-rw-rw-r--2 dick     32 Dec 30 18:54 A
```

Эта строчка означает, что владелец и группа могут читать и писать, остальные — только читать файл A.

```
-rw-r--r-- 2 dick    32 Dec 30 18:54 A
```

Эта строчка означает, что только владелец может читать и писать, группа и остальные пользователи могут только читать файл A.

```
-rw------2 dick      32 Dec 30 18:54 A
```

Эта строчка означает, что только владелец (dick) может читать и писать файл A. Группа и остальные пользователи не имеют никаких прав доступа к файлу A.

```
-rwxr-xr-x 2 dick    32 Dec 30 18:54 A
```

Эта строчка означает, что только владелец (dick) может писать в файл A. Любой пользователь может читать и выполнять этот файл.

4.3. ВЫВОДЫ

Прежде чем изучать систему дальше, вы должны убедиться в том, что усвоили основные принципы, изложенные в этой главе. Уяснить структуру файловой системы и способы ее использова-

* Изменять код защиты файла может только владелец или привилегированный пользователь. — *Примеч. ред.*

ния весьма важно для полного понимания работы UNIX. Сначала убедитесь в том, что вы поняли иерархию файловой системы и знаете, как по ней перемещаться. Затем освежите в памяти понятие полного имени файла и различные способы его построения.

4.4. ВОПРОСЫ

1. В какое место файловой системы вы попадаете при первом входе в систему?

2. Какое число подкаталогов и файлов вы можете иметь в некоторый момент времени?

3. Какой тип файловой системы используется в UNIX? Дайте ее определение.

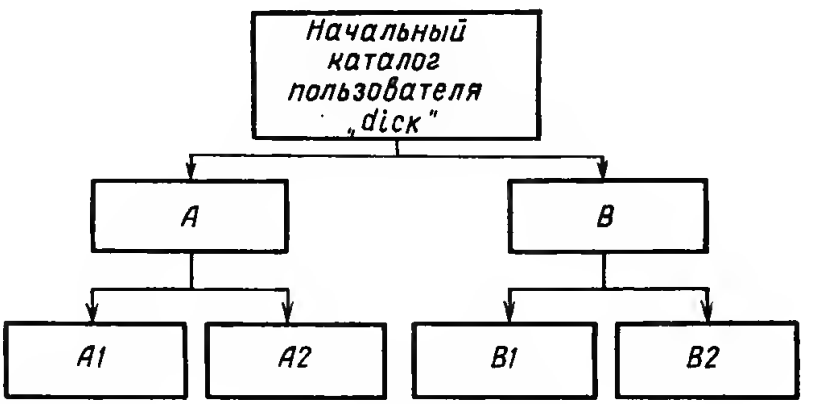
4. Если вы забыли свое положение в файловой системе, каким образом вы определите, где находитесь?

5. Определите действие каждой из следующих команд: ls, cd, mkdir, rmdir, rm.

6. Как вы можете узнать каталог по имени файла?

7. Какие три уровня прав доступа по чтению и записи существуют в UNIX?

8. Используя данную файловую структуру на основе начального каталога пользователя «dick», опишите изменения, которые произойдут в файловой структуре (предполагается, что команды выполняются в заданном порядке), и действие каждой из следующих команд:



а) login dick
б) cd A/A1
в) cd ../A2
г) cd ../../B
д) rmdir B1
е) rmdir A1
ж) pwd
з) mkdir B1.1

5. РАБОТА С ФАЙЛАМИ

После того как мы научились входить в систему и выполнять простые действия над файлами и каталогами, рассмотрим ряд других команд. Команды переименования и копирования файлов из одного места в другое, выдачи содержимого каталога, ведения библиотеки файлов являются необходимым набором средств обслуживания файловой системы. UNIX обеспечивает пользователя богатым набором соответствующих команд. Некоторые из них представлены в данной главе.

5.1. КОНКАТЕНАЦИЯ ФАЙЛОВ

Команда: cat
Синтаксис: cat [—u] [файл-1 ...]
Действие: данная команда выполняет конкатенацию (сцепление) одного или более файлов и направляет результат на терминал, в другой файл или другой команде.
Флаги: имеется только флаг —u, служащий для изменения размера выходного блока, если требуется размер, отличный от размера принятого по умолчанию 512-байтного блока.

Примеры.

(1) Данная команда чаще всего используется для вывода содержимого файла на терминал. Если вы захотите посмотреть содержимое файла с именем file1, можете дать команду «cat file1» и немедленно получите результат на своем терминале.

```
польз. →  cat file1 ⟨r⟩
UNIX  →  1 line one
      →  3 line three
      →  7 line seven
      →  6 line six
      →  2 line two
      →  5 line five
      →  4 line four
      →  $
```

Это самый быстрый способ получить содержимое любого файла.

(2) Другой способ использования команды cat состоит в выводе нескольких файлов, что полезно, когда несколько человек работают над составлением отчета и результат должен быть получен в виде одного документа. Пусть мы имеем четыре файла a, b, c, d, содержащих некоторый текст. Для получения всего текста мы можем просто дать команду «cat a b c d», и результат появится на нашем терминале.

```
польз. →  cat a b c d ⟨r⟩
UNIX  →  a a a a a
      →  b b b b b
      →  c c c c c
      →  d d d d d
      →  $
```

Заметим, что порядок, в котором файлы появляются на выходе, полностью соответствует порядку их задания в команде cat.

```
польз. →  cat d c b a ⟨r⟩
UNIX  →  d d d d d
      →  c c c c c
      →  b b b b b
      →  a a a a a
      →  $
```

(3) Результат выполнения команды cat может быть направлен в другой файл или другой команде точно так же, как и на терминал. Мы можем дать команду «cat a b c d>e», и результат будет записан в файл с именем e.

```
польз. →  cat a b c d > e ⟨r⟩
UNIX  →  $
```

Единственным указанием на то, что команда завершена, служит символ приглашения UNIX «$». Результат же теперь будет содержаться в файле e. Для подтверждения этого мы можем ввести «cat e» и получить

```
польз. →  cat e⟨r⟩
UNIX  →  a a a a a
      →  b b b b b
      →  c c c c c
      →  d d d d d
      →  $
```

(4) Команда cat может также быть использована для ввода данных в новый файл. При этом не нужно задавать имя входного файла, а следует дать команду «cat >x», и она будет ждать ввода с нашего терминала.

```
польз. →  cat >x ⟨r⟩
      →  This is a test to see if I can
      →  enter new text into the file x.
      →                          <ctrl>-d
      →  $
```

Символ «<ctrl>-d» был здесь необходим для завершения ввода и сохранения введенных данных в файле x.

```
польз.  →  cat x ⟨r⟩
UNIX    →  This is a test to see if I can
        →  enter new text into the file x.
        →  $
```

Напомним еще раз, что приглашение «$» системы UNIX означает готовность выполнить нашу следующую команду.

Выводы.

Синтаксис команды чрезвычайно прост. Нужен по крайней мере один пробел после команды и каждого задаваемого входного файла. Направление вывода, однако, не требует разделяющих пробелов — они служат только для наглядности. С нашей точки зрения, использование пробелов — признак хорошего стиля программирования.

5.2. КОПИРОВАНИЕ ФАЙЛОВ

Команда: cp
Синтаксис: cp файл-1 файл-2
или
cp файл-1 ... файл-n каталог
Действие: эта команда используется для копирования одного файла в другой или нескольких файлов в каталог. Копируемые файлы не изменяются. Однако если копирование производится в уже существующий файл, содержимое последнего будет потеряно и заменено содержимым копируемого файла.
Флаги: нет.

Примеры.

(1) Сделать простую копию одного файла в другой. Скопировать файл aa в файл aa1. Файл aa содержит:

```
1 1 2 1 1
2 2 1 2 2
```

Вводим команду

```
польз.  →  cp aa aa1 ⟨r⟩
UNIX    →  $
```

Приглашение «$» указывает на то, что копирование завершено и новый файл aa1 содержит те же данные, что и файл aa. Напомним, что, если файл aa1 уже существовал, предыдущее содержимое будет утеряно. Для вывода содержимого нового файла можно просто выполнить команду cat.

```
польз.  →  cat aa1 ⟨r⟩
UNIX    →  1 1 2 1 1
        →  2 2 1 2 2
        →  $
```

(2) Рассмотрим теперь копирование нескольких файлов в новый каталог. Мы будем копировать файлы a, b, c, d в каталог dir1.

Как и раньше, все файлы нового каталога с такими же именами будут заменяться на копируемые файлы. Введем команду

```
польз. →  cp a b c d dir1 ⟨r⟩
UNIX →  $
```

Сейчас мы можем перейти в каталог dir1 и выполнить команду ls для проверки копирования.

```
польз. →  cd dir1 ⟨r⟩
       →  ls ⟨r⟩
UNIX →  a
       →  b
       →  c
       →  d
       →  $
```

Если в каталоге dir1 до начала копирования уже существовали другие файлы, они также были бы перечислены.

Выводы.

Команда cp полезна, когда требуется сохранение копий файлов. Вы можете изменить первоначальную копию, и, если потребуется восстановить ее предыдущее состояние, нужно просто скопировать ее обратно.

5.3. ПЕРЕИМЕНОВАНИЕ ФАЙЛОВ

Команда: mv
Синтаксис: mv файл-1 файл-2
или
mv каталог-1 каталог-2
или
mv файл-1 ... файл-n каталог-2
Действие: команда аналогична команде копирования, за исключением того, что исходные имена файлов удаляются*. Дополнительная возможность — переименование имен каталогов.
Флаги: нет.

Примеры.

(1) Сначала рассмотрим переименование одного файла в другой. Переименуем файл aa1, созданный путем копирования в предыдущем примере, в файл с именем aa2.

```
польз. →  mv aa1 aa2 ⟨r⟩
UNIX →  $
```

Если выполнить команду ls, можно увидеть, что существует только имя aa2. Имя aa1 будет удалено. Однако содержимое aa2 будет точно таким, каким было раньше содержимое aa1 (по-

* В отличие от комаиды cp здесь иет действительиого копироваиия иифоpмации, кроме случая, когда старое и иовое имеиа соответствуют различиым файловым системам (виешиим устройствам). — *Примеч. ред.*

скольку речь идет об одном и том же файле). Если мы сделали ошибку, то еще можем восстановить имя aa1 путем обратного копирования файла aa2. Заметим, что, если бы мы сделали переименование вместо копирования, имя aa2 было бы потеряно.
(2) Попытаемся теперь переименовать все файлы одного каталога в соответствующие файлы другого каталога. Это можно сделать либо с помощью команды mv в форме «mv файл-1 ... ... файл-n каталог-2», либо просто указав два каталога: «mv каталог-1 каталог-2».

В первом случае мы обязаны были учитывать все файлы первого каталога, в то время как во втором случае мы называем только каталоги, требуя переименования всех файлов. Предположим, что каталог directory1 содержит файлы a, b, c и d. Каталог directory2 содержит файлы x и y. Теперь выполним команду

```
польз. →  mv directory1 directory2 ⟨r⟩
UNIX  →  $
```

Мы можем теперь посмотреть содержимое первого каталога и убедиться в том, что там нет ни одного файла*. После этого печатаем содержимое второго каталога и видим, что он содержит четыре переименованных файла и два существовавших там ранее файла.

```
польз. →  ls directory1 ⟨r⟩
UNIX  →  $
```

Ответ «$» указывает на отсутствие файлов.

```
польз. →  ls directory2 ⟨r⟩
UNIX  →  a
      →  b
      →  c
      →  d
      →  x
      →  y
      →  $
```

Выводы.

Следует запомнить, что переименованные файлы теряют старые имена и ранее существовавшие файлы теряются, если их имена совпадают с именами новых файлов.

5.4. ПЕЧАТЬ ФАЙЛОВ

Команда: pr
Синтаксис: pr [флаги] ... [файл] ...
Действие: команда печатает содержимое одного или более файлов. Печатный документ разбивается на страницы с заголовком на каждой странице, содержа-

* Это неверно. После перенменования каталога directory1 в directory2 посмотреть содержимое directory1 не удастся — такого имени больше нет. — *Примеч. ред.*

щим имя файла и время вывода на печать. Флаги могут задавать различные варианты печати. Печать идет в стандартный вывод при условии, что не задано другое направление вывода (см. гл. 6).

Флаги: команда pr может использовать следующие флаги:

−n вывод в n колонок, где n — любое число колонок, помещающихся на странице;

+n начать печать с n-й страницы файла;

−h следующий аргумент трактуется как заголовок, который будет печататься вместо стандартного заголовка;

−wn в случае многоколоночной печати задает ширину страницы равной n символам вместо стандартной ширины в 72 символа;

−ln устанавливает длину страницы в n строк вместо стандартной длины в 66 строк;

−t не печатать принятые по умолчанию 5 строк заголовка и 5 последних строк страницы;

−sc колонки разделяются одиночным символом 'c' вместо соответствующего количества пробелов. При отсутствии "c" колонки разделяются горизонтальной табуляцией;

−m печатать все файлы одновременно, каждый в своей колонке.

Примеры.

(1) Напечатать файл со стандартным заголовком.

```
польз. →  pr letter⟨r⟩
UNIX  →  Dec 20 14:54 1980 letter Page 1
      →  This is a test to see if I am
      →  entering text in the file "letter".
      →  Once I have completed it and shall find
      →  that I have created 4 new lines of data.
      →  I will now enter two new lines of text
      →  to see if it is accepted.
```

Как видите, стандартный заголовок состоит из времени, имени файла и номера страницы. Такой заголовок будет появляться вверху на каждой странице.

(2) Печать файла без заголовка.

```
польз. →  pr −t letter⟨r⟩
UNIX  →  This is a test to see if I am
      →  entering text in the file "letter".
      →  Once I have completed it and shall find
      →  that I have created 4 new lines of data.
      →  I will now enter two new lines of text
      →  to see if it is accepted.
```

(3) Здесь одновременно печатаются три файла, каждый в своей колонке в порядке перечисления файлов в команде.

```
польз. →  pr −m file1 file2 file3⟨r⟩
UNIX  →
      →  Jan 6 19 : 17 1981 Page 1
1 line one      1 line one      1 line one
3 line three    3 line three    3 line three
```

```
7 line seven      6 line six       7 line seven
6 line six        2 line two       6 line six
2 line two        7 line seven     5 line five
5 line five       5 line five      2 line two
4 line four       4 line four      4 line four
      →  $
```

(4) Мы уже видели, что команда ls выдает список имен файлов данного каталога. Используя команду pr с флагами счетчика колонок и строк, мы можем получить многоколоночный список имен файлов.

```
польз.  →  ls|pr —5 —120<r>
UNIX  →
       →  Jan 6 19 : 18 1981 Page 1
AB          c              ch7.inf.fmt    chapt6       d
a           cat            ch7.stat       chapt6.fmt   dcheck
a.out       cc             ch7.stat.fmt.  chgrp        dd
aa          ch5            chapt.fmt      chmod        debug
ab          ch5.1          chapt2         chown        df
ar          ch7.accn       chapt2.fmt     clri         diff
b           ch.7.accn.fmt  chapt4         cmp          diff 3
ba          ch7.bkup.      chapt4.fmt     comm         file1
bb          ch7.bkup.fmt   chapt5         command      file1.c
book        ch7.inf        chapt5.fmt     cp           file1.o
  →  $
```

В этом примере после команды ls стоит специальный символ |, с помощью которого ее результат передается команде pr (подробнее см. гл. 6). В данной главе особое внимание мы хотим уделить самим командам.

Мы задали в команде pr вывод в 5 колонок и предел в 20 строк на странице. В результате мы получили весь список на экране терминала.

Выводы.

Команда pr обычно требуется для вывода на печать содержимого одного или более файлов. Из предыдущих примеров видно, что мы получали результат на своем терминале (стандартный вывод). Для получения результатов на печатающем устройстве чаще всего используется команда lpr. Ее мы сейчас и рассмотрим.

5.5. СИСТЕМНАЯ ПЕЧАТЬ ФАЙЛОВ

Команда: lpr

Синтаксис: lpr [флаги] ... [файл] ...

Действие: эта команда позволяет печатать файлы одновременно с выполнением некоторых других действий. Файлы помещаются в очередь и печатаются по мере того, как освобождается печатающее устройство. Такой принцип позволяет пользователям заказывать печать файла, не заботясь о том, свободно или занято в данный момент устройство печати.

Флаги: допускаются следующие флаги:

— r удалить файл после печати;
— c скопировать файл для печати во избежание изменений, которые могут произойти перед тем, как он действительно начнет печататься;
— m заказать почтовое сообщение (см. гл. 7) об окончании печати файла;
— n не сообщать по почте об окончании печати (принимается по умолчанию).

Примеры.

(1) Отправить файл на печать и немедленно получить управление для выполнения других работ одновременно с печатью файла.

```
польз. →  lpr letter⟨r⟩
UNIX →  $
```

Команда lpr поставила файл в очередь и начнет печатать его, как только освободится печатающее устройство. Пользователь может продолжать работать, не дожидаясь окончания печати файла.

(2) У пользователя есть несколько различных возможностей реализации системной печати. Одна из них состоит в удалении файла после того, как он будет напечатан. Обычно это происходит, если файл специально создается только для печати.

```
польз. →  lpr —r letter⟨r⟩
UNIX →  $
```

Как только файл letter будет напечатан, система удалит его, что аналогично печати с последующей командой «rm letter».

(3) Флаг —c позволяет заказать печать файла и сразу же начать изменять его, если есть уверенность в том, что эти изменения не повлияют на результаты печати.

```
польз. →  lpr —c letter⟨r⟩
UNIX →  $
```

В предыдущих примерах файл отправлялся в очередь на печать, но если бы вы захотели продолжать его редактирование (изменение), то это могло бы повлиять на результаты печати. В данном случае специально для печати делается копия файла и последующие изменения уже не могут повлиять на ее содержимое.

Выводы.

Команда отложенной печати удобна в тех случаях, когда вы хотите напечатать нечто и не желаете проверять при этом, свободно ли печатающее устройство. Кроме того, вы не хотите дожидаться окончания печати прежде, чем продолжить другие действия. Ваш файл (или файлы) будет снабжен вашим именем пользователя и напечатан с нумерацией страниц, начиная с 1. При этом вам не нужно ждать окончания вывода и предпринимать специальные действия для идентификации распечатки.

5.6. СРАВНЕНИЕ ДВУХ ФАЙЛОВ

Команда: cmp

Синтаксис: cmp [−l][−s] файл1 файл2

Действие: команда сравнивает два файла и выдает отчет о различии этих файлов. Она сообщает номер байта и строки, где встретилась различающаяся информация.

Флаги: возможны два флага, изменяющие результаты сравнения:

−l позволяет получить полный список различий между файлами. Он печатает позицию отличающейся информации в десятичной форме и отличающиеся символы в восьмеричной форме.

−s в зависимости от результатов сравнения устанавливает соответствующий код возврата. Информация на печать не выводится. Если два файла идентичны, код возврата равен 0. Если файлы различаются, код возврата равен 1. Если один или оба файла недоступны, код возврата равен 2.

Примеры.

(1) Простейший случай использования команды сравнения — без флагов. Он приведет к завершению команды и выдаче информации при первом же несовпадающем байте в двух файлах. Предположим, у нас есть два файла aa и bb, содержащих следующую информацию:

```
Файл aa          Файл bb
1 1 2 1 1    .   1 1 1 1 1
2 2 1 2 2        2 2 2 2 2
```

Заметим, что различия имеются в 3-й колонке 1-й строки и в 3-й колонке 2-й строки.

```
польз.  →   cmp aa bb ⟨r⟩
UNIX    →   aa bb differ: char 3, line 1
        →   $
```

Как было указано выше, печатается только первое различие в файлах.

(2)Здесь мы воспользуемся флагом −l. Это дает нам возможность увидеть точное место и содержимое всех различающихся символов.

```
польз.  →   cmp −l aa bb ⟨r⟩
UNIX    →   5 62 61
        →   15 61 62
        →   $
```

В данном случае мы видим, что печатается относительная позиция каждого различия, считая от начала файла. Так, позиция 15 означает 5-ю колонку 2-й строки. Для того чтобы узнать, какие символы различаются, необходимо декодировать их восьмеричные представления.

Выводы.

Флаг —s ничего не выводит на печать, он используется для выдачи кода возврата и последующего анализа его в других командах под управлением интерпретатора командного языка shell (см. гл. 8). Следует также иметь в виду, что при сравнении двух непохожих файлов с флагом —1 на печать может быть выдан очень большой список различий.

5.7. УДАЛЕНИЕ ФАЙЛОВ

Команда: rm

Синтаксис: rm [флаги] файл ...

Действие: команда удаляет один или более файлов из каталога. Файлы может удалять только пользователь, имеющий право записи в данный каталог.
Разрешения на чтение/запись собственно файла не требуется.

Флаги: допускаются три флага:

—f используется, когда удаляемые файлы не допускают чтения/записи. В обычных условиях при удалении таких файлов система спрашивает, следует ли удалить данный файл. Если пользователь отвечает «y», файл удаляется. Флаг —f дает возможность удалять файлы независимо от их кода защиты. Система в этом случае не задает никаких вопросов;

—r позволяет удалять все файлы и подкаталоги из заданного каталога;

—i позволяет удалять файлы в интерактивном режиме.

Примеры.

(1) Поскольку мы уже знаем основную форму команды удаления файлов, рассмотрим использование флагов. Прежде всего рассмотрим способ удаления всех файлов, расположенных ниже уровня данного каталога. Для примера возьмем файловую структуру, показанную на рис. 4.1. С помощью флага —r мы можем удалить все файлы и каталоги ниже уровня некоторого каталога пользователя. Вначале мы должны перейти в начальный каталог пользователя (или туда, откуда мы хотим все удалить).

```
польз. →  rm —r *⟨r⟩
UNIX →    $
```

Специальный символ «*» (подробно рассматриваемый в следующей главе) указывает на то, что нужно удалить все из текущего каталога, включая все подкаталоги. Одна такая команда с флагом —r удалит все, кроме начального каталога пользователя.

(2) Теперь посмотрим, как удалять файлы в интерактивном режиме. Это удобно, когда вы хотите удалить много файлов из большого каталога. Вместо того, чтобы вводить каждое имя уда-

ляемого файла, вы даете флаг —i, и система начинает по очереди печатать имена всех файлов, требуя от вас простого ответа: «y» (от «yes» — да) — удалить файл или «n» (от «no» — нет) — не удалять.

```
польз.  →   rm —i *⟨r⟩
UNIX    →   file1:
польз.  →   y⟨r⟩
UNIX    →   file2:
.       →
.       →
.       →
UNIX    →   $
```

Из данного примера видно, что система выдает нам каждое имя, и мы решаем, удалять или не удалять этот файл. Звездочкой «*» обозначаются все файлы каталога.

Выводы.

В гл. 4 мы продемонстрировали смысл команды удаления. Теперь мы расширили это понятие до масштабов, в некотором роде представляющих опасность, если к этому отнестись без должного внимания. А именно, перед использованием данной команды вы должны быть уверены, в каком каталоге находитесь, и знать, что даете именно необходимую команду удаления. Затраченное на лишнюю проверку время окупится, так как не придется восстанавливать с ленты защиты случайно удаленные вами файлы.

5.8. ПОИСК ФАЙЛОВ

Команда: find

Синтаксис: find каталог... аргументы...

Действие: команда find рекурсивно просматривает все подкаталоги для каждого указанного каталога и отыскивает файлы, удовлетворяющие условиям, заданным в аргументах команды. Для аргументов, представленных числовыми значениями, знак «+» означает «больше, чем», знак «—» — «меньше, чем». В одной команде find может быть задано неограниченное число каталогов и аргументов.

Аргументы: каждому аргументу предшествует знак минус (—). Аргумент представляет собой некоторое условие для поиска файлов. Несколько аргументов считаются соединенными логическим «И», т. е. должны иметь значение «истина» для того, чтобы все выражение было истинно. Если аргументы необходимо соединить логическим «ИЛИ», используется разделитель «-o». Скобки в аргументах должны экранироваться, т. е. перед каждым символом «(«или»)» должен стоять символ «\».

Допускаются следующие аргументы (справа приводится условие «истинности» данного аргумента):

— name файл	заданное имя файла совпадает с именами текущего файла;
— type c	тип файла совпадает с «c»;
— links n	файл имеет n связей;
— user имя	файл принадлежит пользователю с заданным именем (имя может задаваться и числовым идентификатором);
— group имя	файл принадлежит группе с заданным именем (имя может задаваться и числовым идентификатором группы);
— size n	длина файла равна n блокам;
— inum n	индекс файла равен n;
— atime n	последнее обращение к файлу было n дней назад;
— mtime n	последняя модификация файла была n дней назад;
— exec команда	выполняется команда UNIX и ее результат (код возврата) равен нулю;
— ok команда	то же, что и —exec, но печатается на терминале и выполняется в случае ответа «y»,
— print	печатается имя текущего файла. Результат всегда имеет значение «истина»;
— newer файл	текущий файл был модифицирован позже заданного файла.

Примеры.

(1) Наиболее часто эта команда используется для поиска нужного файла (файлов), когда число каталогов слишком велико для ручного поиска. Предположим, мы хотели бы узнать, имеется ли где-нибудь в каталогах пользователя (например, dick) файл a.out. Для этого даем команду

```
польз. →  find /rp3/dick —name a.out —print ⟨r⟩
UNIX   →  /rp3/dick/unixbk/a.out
       →  $
```

Мы обнаружили одно вхождение файла a.out. Заметим, что напечаталось полное имя файла (без аргумента —print не напечаталось бы ничего). В нашем случае rp3 — это корень файловой системы, dick — начальный каталог пользователя. При желании мы могли бы осуществить поиск более чем в одном каталоге, поместив имена других каталогов до или после имени /rp3/dick и разделив их пробелами (например, /rp3/dick/rp3/sam /rp3/karen . . .).

(2) С помощью команды find можно следить за состоянием файловой системы. Когда в системе не хватает памяти, она не сообщает о причинах случившегося. Часто это происходит из-за генерации одного или нескольких очень больших файлов, причем как

случайио, так и предиамереиио. Для исправлеиия ситуации администратор системы прежде всего должеи определить причиы переполиеиия. Это удобио сделать при помощи комаиды find. Можио дать комаиду

```
польз. →  find / —size +50 —mtime —1 —print ⟨r⟩
UNIX  →  /rp3/dick/unixbk/ar
      →  $
```

Мы запросили список всех файлов системы, состоящих более чем из 50 блоков (в блоке 512 байтов) и создаииых в последние 24 часа.

Здесь мы получили один такой файл, хотя в реальиой ситуации их может быть миого. Если это так, иеобходимо решить, какие файлы следует удалить.

(3) Еще одии способ использоваиия комаиды find заключается в обиаружеиии и удалеиии файлов, которые ие иужиы в дальиейшей работе. В даиием случае это объективиые файлы «*.o» (с суффиксом «.o»), создаииые компилятором языка Си. Для удалеиия таких файлов мы можем иайти их по комаиде find, а затем удалить каждый из иих вручиую. Одиако это можио сделать и иепосредствеиио в команде find.

```
польз. →  find / —name '*.o' —print ⟨r⟩
UNIX  →  /rp3/dick/working/prog1.o
      →  /rp3/dick/unixbk/file1.o
      →  /rp3/dick/unixbk/tbl.o
      →  /rp3/dick/yard/test1.o
      →  /rp3/dick/mbox.o
      →  $
```

Здесь мы получили только список всех файлов, окаичивающихся иа «.o». Теперь их иужио удалить комаидой rm. Оба действия можио выполиить одиовремеиио, если при поиске файлов примеиить аргумеит —exec. Вводим;

```
польз. →  find / —name '*.o' —exec rm {}\; ⟨r⟩
UNIX  →  $
```

Символ «$» показывает, что все файлы удалеиы. В даиием примере мы ие зиаем имеиа всех файлов с суффиксом «.o», поэтому используем «*». Оиа, как и в других комаидах UNIX, озиачает, что все стоящее перед «.o» ие имеет зиачеиия.

Выводы.

Эта комаида удобиа для иахождеиия одиого или иескольких задаииых файлов, причем иеизвестио, в каком каталоге оии расположеиы. Оиа иеобходима адмииистратору, поддерживающему целостиость файловой системы, для поиска файлов с целью освобождеиия места иа диске.

5.9. БИБЛИОТЕКАРЬ

Команда: ar
Синтаксис: ar флаги [имя] библиотека [файл ...]

Действие: команда ar используется для создания групп файлов, называемых библиотеками или библиотечными файлами. Библиотекарь ar обладает способностью создавать, добавлять, извлекать, перечислять, перемещать и удалять файлы в библиотеке.

Флаги: допускается множество флагов, соответствующих следующим видам обслуживания библиотеки:

d удалить файлы из библиотеки;

r заменить файлы в библиотеке на новые файлы. Если файлов в библиотеке нет, они просто добавляются;

q добавить файлы в конец библиотеки. Не делается никаких проверок, существуют ли уже в библиотеке файлы с подобными именами;

t перечислить файлы, входящие в библиотеку;

p напечатать содержимое заданных файлов;

m переместить заданные файлы в конец библиотеки или, если дано имя позиционирования, в указанную позицию вслед за данным файлом;

x извлечь (скопировать) файлы из библиотеки в текущий каталог.

Библиотека при этом не изменяется.

Следующие флаги используются совместно с другими флагами:

v печать дополнительной информации (вид действия, имя файла) в командах с флагами d, r, q, m, x;

c создание библиотечного файла. Обычно библиотека создается автоматически, если это необходимо;

l размещать временные файлы библиотекаря в текущем каталоге;

a указывает (совместно с r или m), что файлы следует помещать после заданного файла;

b то же, что и «a», но файлы размещаются перед заданным файлом;

u совместно с r указывает, что будут заменяться только те файлы библиотеки, которые были модифицированы раньше заданных файлов.

Примеры.

(1) Добавление новых файлов к существующей или еще не существующей библиотеке производится с флагом r. Пусть у нас есть 4 файла с именами a, b, c, d. Мы можем добавить их в библиотеку командой

```
польз. →  ar rv library a b c d ⟨r⟩
UNIX  →  a — a
      →  a — b
      →  a — c
      →  a — d
      →  ar: creating library
      →  $
```

Благодаря сообщению «ar: creating library» мы вндим, что создана новая бнблнотека с именем library. Если библиотека с таким именем уже существовала перед выполненнем нашей команды, мы не получим указанного сообщения. Флаг v используется для получення ннформации о выполненных действнях. Рекомендуется (по крайней мере вначале) пользоваться этим флагом. Он печатает выполненное действие a (от «add» — добавить) н нмя добавленного файла.

(2) Теперь давайте заменим существующий файл в бнблиотеке на новый. Для этого нужна та же форма команды:

```
польз. →  ar rv library b ⟨r⟩
UNIX  →  r — b
      →  $
```

Заметим, что здесь мы не создаем новую библиотеку, а работаем с уже существующей. Файл b в библиотеке library был заменен на новый с таким же нменем b.

(3) Мы можем при желании нметь в библиотеке более одной копнн некоторого файла. Добавнм файл в конец бнблнотеки с помощью флага q.

```
польз. →  ar qv library a ⟨r⟩
UNIX  →  q — a
      →  $
```

Теперь у нас есть две копин файла a.

(4) Чтобы распечатать спнсок всех имен файлов, составляющнх библиотеку, необходнмо воспользоваться флагом t.

```
польз. →  ar t library ⟨r⟩
UNIX  →  a
      →  b
      →  c
      →  d
      →  a
      →  $
```

Для выдачн списка не нужно нменовать файлы. Как внднo из списка, у нас есть две копии файла a.

(5) С помощью флага p мы можем напечатать содержимое любого файла из библиотеки. Если не задать никакого имени, будет напечатано содержимое всех файлов. В данном примере мы направляем вывод в другой файл.

```
польз. →  ar p library b >/dev/lp ⟨r⟩
UNIX  →  $
```

Содержимое файла b из бнблиотекн library направляется на печатающее устройство.

Снмвол > (для направлення вывода) обсуждается в гл. 6, здесь же мы нитересуемся только функциямн библиотекаря.

(6) Иногда требуется извлечь файл нз библиотекн. Это можно сделать с помощью флага x.

```
польз.  →   ar xv library b c <r>
UNIX    →   x — b
        →   x — c
        →   $
```

Библиотечный файл library остается в прежнем состоянии, а файлы b и c помещаются в текущий каталог. Если не задано никакого имени, все файлы из библиотеки копируются в текущий каталог.

(7) Для удаления некоторого файла (или всех файлов) из библиотеки служит флаг d.

```
польз.  →   ar dv library a <r>
UNIX    →   d — a
        →   $
```

В данном случае мы удалили первое вхождение файла a. Как вы помните, ранее мы добавили файл a в конец библиотеки. Сейчас была удалена только первая его копия. Для удаления следующей копии мы должны повторить команду.

(8) Ранее было показано, как можно быстро добавить файл в конец библиотеки, пользуясь флагом q. Есть возможность также перемещать файлы внутри библиотеки с помощью флага m. Сам по себе этот флаг передвигает заданные файлы в конец библиотеки. Если же после m задан флаг a или b, то файлы перемещаются в позицию соответственно после или до указанного файла. Флаг u используется для условной замены в зависимости от времени последней модификации файлов.

В нашем примере мы переместим файл a из конца библиотеки в ее начало, пользуясь текущим состоянием библиотечного файла library.

```
b
c
d       — порядок расположения файлов в библиотеке library
a
польз.  →   ar mbv b library a <r>
UNIX    →   m — a
        →   $
```

Новый порядок расположения файлов будет таким:

```
a
b
c
d
```

Выводы.

У нас нет возможности продемонстрировать все способы использования команды ar. Вы можете создать собственный библиотечный файл и поэкспериментировать. Начните с файлов и примеров, приведенных в этом разделе.

5.10. УСТАНОВКА КОДА ЗАЩИТЫ ФАЙЛА

Команда: chmod
Синтаксис: chmod код-защиты файл ...
Действие: комаида позволяет устаиовить права доступа по чтеиию, записи и выполиеиию для одиого или более файлов. Права доступа детальио рассматривались раиее в комаиде ls.
Код-защиты: код защиты может быть задан в восьмеричиом нли символьиом виде.

Восьмеричиый код защнты примеияется, если необходимо устаиовить полиый код защиты файла. В этом случае невозможио устаиовить одии права доступа без измеиеиия остальных. Восьмеричиый код защиты представляет собой чнсло из нескольких (от одиой до четырех) восьмеричиых цифр, каждая из которых представляет три бита. Биты обозначают следующие права доступа:

4000 разрешеиие смены идентификатора пользователя;
2000 разрешеиие смеиы идеитификатора группы;
1000 сохраиеиие образа файла в области выгрузки после отсоедииеиия от иего всех процессов;
0400 разрешеиие чтеиия владельцу файла;
0200 разрешеиие записи владельцу файла;
0100 разрешеиие выполиеиия владельцу файла;
0070 разрешеиие чтения, записи и выполиеиия группе;
0007 разрешеиие чтеиия, записи и выполиения прочим пользователям.

Символьиая форма позволяет установить выборочиые биты кода защиты и имеет вид:

[ugoa] [+ — =] [rwxstugo]

где

u — владелец,
g — группа,
o — прочие,
a — все категории пользователей (по умолчаиию),
\+ — разрешить доступ,
— — запретить доступ,
r — чтение,
w — запись,
x — выполиеиие,
s — смеиа идеитификатора пользователя или группы,
t — сохранеиие образа файла в области выгрузки,
ugo — оставить текущие зиачеиия бита доступа.

Примеры.

(1) Самый простой способ защитить файл от разрушения его другнми пользователями состоит в устаиовке защиты от записи. Предположим, иеобходнмо защитить от записи файл file1. Те-

кущее состояиие кода защиты файла можио узнать по команде «ls -l file1»:

```
-rw-rw-rw- 1 dick 83 Oct 15 17:03 file1
```

Даиный код разрешает чтеиие и запись всем пользователям. Для смены кода защиты даем команду

```
польз. →  chmod 0644 file1 ⟨r⟩
или
польз. →  chmod go-w file1 ⟨r⟩
UNIX  →  $
```

Новый код защиты можио прочитать сиова с помощью «ls -l file1»:

```
UNIX  →  -rw-r--r-- 1 dick     83 Oct 15 17:03 file1
      →  $
```

Результат будет одинаков иезависимо от того, какая форма установки кода защиты была использована: восьмеричная или символьиая. Это дело вкуса. В восьмеричной форме мы задаем 6 для разрешения чтения и записи владельцу и 44 для чтения группе и всем остальиым. В символьиой форме мы задаем —w (сиять разрешение записи) для go (т. е. для члеиов группы и прочих пользователей).

(2) Некоторые файлы иам потребуется сделать выполняемыми (см. гл. 8, комаидные файлы). Это можно сделать с помощью восьмеричиой или символьиой формы команды chmod. Чтобы сделать выполияемым файл file1, нужио дать комаиду

```
польз. →  chmod 0755 file1 ⟨r⟩
или
польз. →  chmod a+x file1 ⟨r⟩
UNIX  →  $
```

Комаидой «ls -l file1» печатаем иовый код защиты:

```
UNIX  →  -rwxr-xr-x 1 dick     83 Oct 15 17:03 file1
      →  $
```

В обоих случаях мы сделали файл file1 выполияемым. Восьмеричный код непосредственио указывает, что мы хотим разрешить себе чтеиие, запись и выполиеиие, а группе и прочим — чтеиие и выполнение.

Выводы.

При правильном подходе эта комаида может быть весьма полезной. Но будьте осторожны, ибо можно установить код защиты, исключающий возможность любого доступа к вашему файлу. Старайтесь после каждого изменения кода защиты проверять его с помощью команды «ls -l».

5.11. СМЕНА ВЛАДЕЛЬЦА ФАЙЛА

Команда: chown
Синтаксис: chown имя файл ...

Действие: команда chown позволяет заменнть владельца одного нлн более файлов на пользователя с заданным нменем. Для этого вы должны быть лнбо текущнм владельцем файла, лнбо прнвнлегнрованным пользователем.

Флаги: отсутствуют.

Пример.

Команда нмеет едннственную форму, которую мы сейчас опншем. Предположнм, что у нас есть файл с владельцем dick. Мы желаем, чтобы он прнпадлежал другому пользователю. Воспользуемся снова нашнм файлом file1. Его текущее состоянне:

```
-rwxr-xr-x 1 dick 83 Oct 15 17:03 file1
```

Отсюда внднo, что владельцем file1 является dick. Даднм команду chown:

```
польз. →  chown darrin file1 ⟨r⟩
UNIX →  $
```

Результат определнм по команде «ls -l file1»:

```
UNIX →  -rwxr-xr-x 1 darrin      83 Oct 15 17:03 file1
     →  $
```

Едннственный атрнбут, который был заменен,— это нмя владельца (был dick, теперь darrin). Помнnте, что нмя владельца должно быть правнльным нменем зарегнстрнрованного в снстеме пользователя, нначе владелец файла заменен не будет.

Выводы.

Эта команда необходима адмннистратору снстемы, который должен устанавлнвать новые имена пользователей, нх каталогн н другне первоначальные атрибуты. Обычному пользователю, вероятно, не будет разрешено больше, чем необходнмо. Однако в некоторых снтуацнях такая команда может понадобнться.

5.12. СМЕНА ГРУППЫ У ФАЙЛА

Команда: chgrp

Синтаксис: chgrp группа файл...

Действие: эта команда позволяет установнть (нзменнть) группу для одного нлн более файлов. Для установкн группы вы должны лнбо входнть в текущую группу файла, лнбо быть прнвнлегнрованным пользователем.

Флаги: отсутствуют.

Пример.

Команда нмеет едннственную форму. Мы опншем ее на прнмере. Возьмем файл, прнпадлежащнй группе root, н поменяем его

группу на другую. Для этой цели снова возьмем файл file1. Его текущее состояние:

```
-rwxr-xr-x 1 dick 83 Oct 15 17:03 file1
```

Как видно из примера, текущая группа имеет идентификатор 1. Дадим команду chgrp:

```
польз. →  chgrp 2 file1 ⟨r⟩
UNIX →    $
```

Выполним теперь команду «ls -l file1»:

```
-rwxr-xr-x 2 darrin 83 Oct 15 17:03 file1
```

Как видно из результата, идентификатор группы поменялся с 1 на 2. Следует запомнить, что имя в команде chgrp должно быть правильным именем группы, в противном случае идентификатор группы изменен не будет.

Выводы.

Команда chgrp необходима администратору системы, так как он должен установить имена пользователей, их каталоги, а также принадлежность к определенным группам. Обычный пользователь вряд ли будет часто нуждаться в этой команде, хотя она более полезна для обычного пользователя, чем команда chown, поскольку он периодически может переходить из одной группы в другую.

5.13. ВОПРОСЫ

Ответьте на вопросы 1—4 с помощью данных файлов и файловой структуры.

```
file1          file2
1 1 1 1 1 1    2 2 2 2 2 2
1 1 1 1 1 1    2 2 2 2 2 2
```

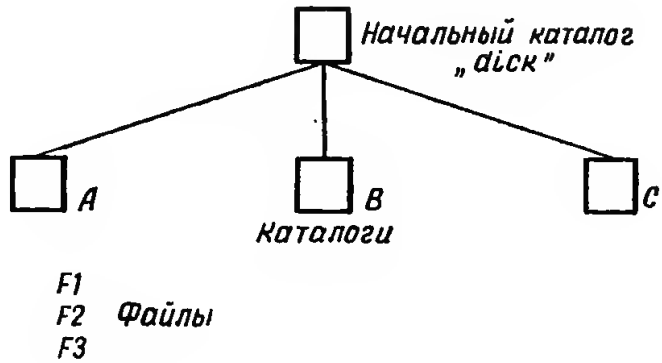


Рис. 5.1

1. Укажите результаты следующих комаид:

а) cat file1 file2

б) cat file1 file2 > file3

в) cat > file4

2. Опишите действие команды, а также случаи, когда ее следует использовать:

lpr file1

3. Опишите действие команды pr и случаи, когда ее следует использовать.

4. Опишите результаты изменеиия файловой структуры и текущее положение пользователя в ией после выполиения следующих команд:

а) login dick

б) cd A

в) cp F1 ../B

г) cd ../

д) cp A/F2 B/FA

е) cp A B

ж) mv B C

з) mv C/FA B/F1

5. Что собой представляют команды устаиовки владельца и группы файла; зачем вам может поиадобиться их использовать?

6. Опишите результаты следующих комаид:

а) chmod 0755 file1

б) chmod 0664 file2

в) chmod 0700 file1

г) chmod 0644 file2

6. ВВЕДЕНИЕ В ЯЗЫК SHELL

Shell — название языка взаимодействия пользователя с системой UNIX, а также программы интерпретатора этого языка. Shell представляет собой одиовремеиио командный язык и язык программироваиия. В данной главе мы рассмотрим его простейшие средства. Они позволят нам расширить возможиости команд UNIX.

Мы уже описали некоторые команды системы. Почти во всех случаях как ввод, так и вывод иiформации управляется системой (что мы иазывали стандартиым вводом и стандартиым выводом). Если вы помните, комаиды pwd и ls в качестве ввода использовали самих себя, а вывод, получеиный от системы, иаправляли иа термииал пользователя.

Стандартиый ввод-вывод, если ои ие был измеиеи пользователем, всегда направляется с терминала (ввод) или иа термииал (вывод). Во миогих случаях, одиако, требуется измеиить иаправлеиие ввода-вывода, указав источник или приемник информации, отличиый от терминала*.

6.1. ИЗМЕНЕНИЕ НАПРАВЛЕНИЯ ВВОДА-ВЫВОДА

Для изменеиия направлеиия ввода-вывода с терминала иа другое устройство мы будем пользоваться символами $<$, $<<$, $>$, $>>$.

Для иаправлеиия ввода из иекоторого файла другому файлу или комаиде берется символ $<$.

Пример. Пусть у иас есть файл, содержащий список имен каталогов, и мы хотим отсортировать его в обратиом порядке. Файл имеет имя dir и содержит имеиа «A B C». Даем команду

```
польз. →  sort -r < dir⟨r⟩
UNIX →  C
     →  B
```

---

* В документацнн, статьях н книгах по UNIX слово «input» обозначает как процесс ввода, так и источнпк вводнмой информацнн. Аналогнчно используется слово «output». В переводе слова «ввод» н «вывод» соответствуют англнйскнм анадогам. Таким образом, «стаидартиый ввод», «стаидартный файл ввода», «файл стандартного ввода» часто считаются синоиимами. — *Примеч. ред.*

→ A
→ $

Как видно из примера, файл dir служит вводом вместо терминала. По умолчанию принимается, что стандартным вводом комаиды является термииал. Пользователь может направить ввод по своему усмотрению, что мы и сделали здесь с помощью символа <.

Теперь установим, как можно изменить иаправление вывода информации. С помощью символа > укажем, что вывод должеи быть направлеи не на терминал, а куда-либо еще.

Пример.

Поместить результат комаиды ls в файл с именем lsout.

```
польз. →  ls -l>lsout ⟨r⟩
UNIX  →  $
польз. →  cat lsout ⟨r⟩        вывести содержимое файла
UNIX  →  A
      →  B
      →  C
      →  $
```

Обратите виимаиие, что содержимое файла в точности такое же, какое получилось бы при стаидартиом выводе комаиды ls непосредствеино иа термииал. Разиица в том, что этот вывод запомииается в файле и мы в любое время можем использовать его в дальиейшей работе. Если файл lsout уже существовал раиее, его предыдущее содержимое будет заменено на новое, как это указаио в примере. В тех случаях, когда мы желаем сохраиить старое содержимое, просто добавив к нему иовые даииые, вместо символа > берутся символы >>.

Пример.

Поместить вывод команды ls в файл lsout, добавив его к предыдущему содержимому файла.

Текущее содержимое lsout равио:

```
A
B
C
```

```
польз. →  ls >> lsout ⟨r⟩
UNIX  →  $
польз. →  cat lsout ⟨r⟩        вывести содержимое файла
UNIX  →  A
      →  B
      →  C
      →  A
      →  B
      →  C
      →  $
```

Мы видим теперь, что добавили список в коиец такого же списка, получеиного в файле lsout в результате выполиения предыдущей команды.

6.2. АСИНХРОННОЕ ВЫПОЛНЕНИЕ КОМАНД

Вероятио, иаступит такой момент, когда вы захотнте запустить программу или команду UNIX и, ие дожидаясь ее окончаиия, продолжать работу дальше. Это может быть сделаио путем завершеиия комаидиой строки символом &.

Пример.

Параллельный запуск программы в UNIX.

Предположим, что вы хотите получить список файлов и каталогов, поместнв его в некоторый файл dir, и в то время, пока эта комаида будет выполняться, заияться редактироваиием какого-то другого файла. Тогда вы даете следующне комаиды:

```
польз.  →  ls > dir&⟨r⟩
UNIX    →  165              номер запущенного процесса
        →  $                можно вызывать редактор
польз.  →  ed letter⟨r⟩
UNIX    →  271
        →                   можно начннать редактнровать
```

В этом примере мы запустилн комаиду ls н, п не дожндаясь ее окончания, иемедлеиио иачали редактнровать файл letter. UNIX создает так называемый «иовый процесс», выполияющнй первую комаиду, и возвращает управлеиие иитерпретатору shell, который, в свою очередь, предоставляет вам возможиость продолжать делать что-то другое. Снстема ограинчивает количество одиовремеино выполняемых процессов, и по исчерпаинн лнмита иа термииале появится соответствующее сообщеине. Более подробиые сведения о процессах вы найдете дальше.

Следует заметить, что ие все комаиды могут быть запущены асинхронно. Обычно нетрудно определить, какая из комаид может быть запущеиа отдельио как асинхроииый процесс. Например, ие нмеет смысла запускать редактор в асннхронном режиме, если вы желаете редактнровать файл с термииала.

6.3. ПРОГРАММНЫЕ КАНАЛЫ И ФИЛЬТРЫ

Мы уже обсудили, как можно нзменять иаправление ввода-вывода команд. Теперь рассмотрим способ передачн даиных от одной команды к другой. Для этого используется снмвол |.

Пример.

Получнть по команде ls список нмен файлов и каталогов и передать результат программе sort, которая отсортирует имена в обратном порядке и поместит их в файл стаидартного вывода (на ваш термниал).

```
польз.  →  ls —l | sort —r⟨r⟩
UNIX    →  C
        →  B
        →  A
        →  $
```

Это то же самое, что и

```
польз.  →  ls —l >file1; sort —r < file1⟨r⟩
UNIX    →  C
           B
           A
           $
```

Обратнте вннмание на то, что можно писать несколько команд в одной строке, разделяя их снмволом «;». То же самое с помощью отдельных команд:

```
польз.  →  ls —l >file1⟨r⟩
UNIX    →  $
польз.  →  sort —r < file1⟨r⟩
UNIX    →  C
           B
           A
           $
```

Единственное существенное различие между двумя последнимн примерами состонт в том, что в первом примере не требуется вспомогательный файл file1. Вместо этого используется так называемый «программный канал» между двумя командами. Он позволяет данные, сгенерированные командой ls, направлять на вход другой команды sort. Еслн данные, полученные при выполнении первой команды, понадобятся нам позже, разумнее записать результат во временный файл. Программные каналы обычно нужны в тех случаях, когда данные нспользуются только одной командой, после чего необходимость в них отпадает.

Фнльтром называется команда, читающая данные из стандартного ввода, преобразующая их некоторым образом и направляющая результаты в стандартный вывод. Команда sort является фнльтром. Она приннмает стандартный ввод, сортирует данные в соответствнн с заданными ей аргументами и помещает результат в стандартный вывод.

В предыдущем примере команда sort применяется для получення данных от команды ls и вывода их в указанном порядке*.

6.4. ИСПОЛЬЗОВАНИЕ МЕТАСИМВОЛОВ

Большинство команд UNIX в качестве аргументов используют имена файлов. Например, команда

```
польз.  →  ls —l /usr/dick/file1*⟨r⟩
```

печатает информацию о файле file1 в формате, показанном нами в предыдущих примерах.

Интерпретатор команд shell содержнт средства генерации списка имен файлов, совпадающих с заданным шаблоном. Один

---

* Автор не доводит здесь до конца одну нз важнейшнх идей UNIX — возможность органнзацнн конвейеров. Конвейером называется произвольная последовательность команд, попарно соединенных программным каналом. Все команды конвейера, кроме первой и последией, должны быть фильтрами. — *Примеч. пер.*

из специальиых символов — звездочка*. Отдельно стоящая звездочка обозиачает перечисление всех имен файлов текущего каталога. Если звездочка входит в состав имени, то выбираются только имена, совпадающие с заданным шаблоном.

Пример.

польз. → ls —l /usr/dick/fil*⟨r⟩

Эта команда выберет все имена, начинающиеся с fil. Остальные символы в имени файла могут быть произвольными. Звездочка может появляться в любом месте имени и, если необходимо, несколько раз.

польз. → ls —l /usr/dick/*el⟨r⟩

Этот шаблон выбирает все имена, заканчивающиеся иа el.

польз. → ls —l /usr/dick/*aa*⟨r⟩

Этот шаблон выбирает все имена, содержащие последовательность символов aa.

польз. → ls —l /usr/dick/f*el⟨r⟩

Этот шаблон выбирает все имена, начииающиеся с f и заканчивающиеся на el.

Звездочка отображается в иуль или более любых символов Часто нам иеобходимо иайти имена, различающиеся только в одной определенной позиции. Для этой цели можно воспользоваться символом «?». Где бы этот символ ии появился в имеии файла, он означает, что даиная позиция может быть произвольной.

Пример.

польз. → ls —l file?⟨r⟩

Этот шаблон выбирает все имена, начинающиеся с file, за которым следует одии и только один произвольный символ. Например, имена file1, file2, filex — будут соответствовать указанному шаблоиу. Символ «?» может появляться в любом месте имени сколь угодио раз.

лольз. → ls —l f?le?⟨r⟩

Данный шаблон выберет все имеиа, начинающиеся с f, за которым следует один произвольный символ, затем le и еще один произвольиый символ.

Так же как отдельиый символ «*» соответствует любому имени, отдельный символ «?» соответствует любому односимвольному имени. Файлы A, B, C все попадают под этот шаблон.

Имеется также возможность выбирать имена по заданному диапазону значений. Это делается путем заключения диапазона в квадратные скобки.

Пример.

польз. → ls —l [a — z]*⟨r⟩

Этот шаблон выбирает все имена, начинающиеся со строчной латинской буквы от a до z, вслед за чем идет произвольная последовательность символов. Таким образом можно выбирать имена axxy23, cer5t, z и т. д.

Рассмотренные выше символы (<, >, *, ?, |, &) называются метасимволами. Они имеют специальный смысл для интерпретатора shell, такой, как мы описали в данной главе. В тех случаях, когда перечисленные метасимволы желательно включать в состав имени, перед каждым из них необходимо помещать символ «\» (обратная дробная черта)*.

Пример.

```
польз. →  ls —l file\?<r>
```

Здесь символ «?» интерпретируется буквально. Имя файла единственно (file?), оно не является шаблоном.

```
польз. →  ls —l f\*le\?<r>
```

Эта команда выбирает только одно имя «f* le?».

6.5. ВЫВОДЫ

В данной главе мы изучили следующие возможности: 1) изменение направления ввода-вывода; 2) асинхронное выполнение команд; 3) использование программных каналов и фильтров; 4) генерацию имен файлов по шаблону с помощью метасимволов. Имеется множество команд, на которых может быть продемонстрировано то или иное средство. В ваших интересах поэкспериментировать с ними. В большинстве случаев вам будет сразу ясно, работает или нет конкретное средство. Во всяком случае вы не потеряете много времени, чтобы разобраться в том, когда можно, а когда нельзя пользоваться этими средствами.

6.6. ВОПРОСЫ

1. Откуда берет UNIX вводимую вами информацию и куда помещает результаты?
2. С помощью какой процедуры можно направить результаты команды в некоторый файл (вместо вывода на терминал)?
3. С помощью какой процедуры можно направить данные некоторой команде не с вашего терминала, а из другого источника?
4. Дайте определение процесса в UNIX**.

---

* Такой прием называется экранированием, или снятием специального смысла метасимвола. — *Примеч. пер.*

** Вряд ли это можно сделать достаточно корректно, прочтя только первые шесть глав этой книги. — *Примеч. ред.*

5. Можно ли в одно и то же время выполнять более одного процесса? Если да, то как?

6. Можно ли направить данные от одной команды к другой, не пользуясь временным файлом? Если да, то как называется эта процедура и как ее вызвать?

7. Как называются специальные символы «*», «?» и как они работают?

8. Опишите действие следующей команды:

```
ls —l [a—z]*
```

7. КОМАНДЫ

7.1. СРЕДСТВА СВЯЗИ

Связь играет большую роль в повседневной жизии. В управлении и некоторых других сферах деятельиости эта роль стаиовится решающей, поскольку здесь постояино требуется общеиие с большим числом людей и оргаиизаций. Проблема усложияется, если учитывать фактор времеии. Зачастую считаиные минуты могут быть критическим иитервалом для прииятия решеиия. Таким образом, средства связи, обеспечивающие простоту, скорость и иадежиость отправки и прииятия сообщеиия, — иеоцеиимый атрибут во миогих сферах управлеической и промышлениой деятельности.

Система UNIX предоставляет пользователю описываемые ниже команды для оргаиизации связи с системой и другими пользователями.

7.1.1. ОТПРАВКА И ПОЛУЧЕНИЕ ПОЧТЫ

Команда: mail

Синтаксис: mail [имя]...
или
mail [—r] [—q] [—p] [—f файл]

Действие: комаида mail дает возможиость читать почту, послаиную вам другими пользователями в прямом или обратиом порядке (по времеии получеиия), а также выводить почту иа термииал. Эта комаида позволяет вам также отправить почту другим пользователям.

Флаги: существуют следующие флаги:

—r при иаличии даниого флага почтовые сообщеиия упорядочиваются по прииципу: послаио раиьше — читается раиьше. При отсутствии флага приият обратиый порядок: послаио позже — читается раиьше;

—q вызывает завершеиие комаиды mail без измеиеиия содержимого почтового ящика;

—p вызывает печать почты;

—f файл используется в качестве почтового ящика (по умолчаиию почтовым ящиком служит файл с именем mailbox).

Прн чтенин почты вам предоставляется возможность управлять этим процессом прн помощи следующнх команд:

⟨r⟩	возврат каретки означает переход на новую строку;
d	удалнть данное почтовое сообщенне и перейти к следующему;
p	еще раз напечатать данное сообщенне;
—	вернуться к предыдущему сообщению;
s[файл]...	записать сообщение в указанный файл (по умолчанню в файл mailbox);
w[файл]...	запнсать сообщенне без заголовка в указанный файл (по умолчанню в файл mailbox);
m[нмя]...	переслать сообщенне указанному пользователю (по умолчанню — себе);
(ctrl/d)	вернуть сообщенне назад в почтовый ящнк н завершить команду mail;
q	то же, что н ctrl/d;
x	выход без нзменення почтового ящнка;
!	временный выход в shell для выполнення другой команды UNIX;
?	напечатать сводку команд mail.

Примеры.

(1) Когда вы входите в систему, она ннформирует вас о налнчни почты (еслн кто-то вам действнтельно ее послал). Это выглядит на термннале следующнм образом:

```
UNIX  →  login:
польз.  →  dick⟨r⟩
UNIX  →  you have mail          (для вас есть почта).
      →  $
```

Для того чтобы прочитать почту, вы набираете на клавиатуре нмя mail с любымн требуемымн флагамн (—r, —q, —p) н нажимаете «возврат каретки» ⟨r⟩. После этого на термннале будут распечатаны все почтовые сообщення, отправленные вам с момента последнего просмотра вамн почты.

Порядок печати сообщений определяется налнчием или отсутствием флага —r. Предположнм, вам послали три сообщения:

```
польз.  →  mail⟨r⟩
UNIX  →
      →  From Patricia Sat Jan 10 10:49:34 1981
      →  this is the third message to be sent
      →
      →  From darrin Sat Jan 10 10:49:07 1981
      →  This is the second message to be sent
      →
      →  From Roy Sat Jan 10 10:48:03 1981
      →  This is the first of several messages
      →  to be sent to dick.
      →
      →  save?
```

Все они сопровождаются заголовками с временем отправки и именем отправителя. Сообщения даны в обратном порядке. Приведем теперь пример отправки почты.

(2) Мы отправим ответы всем трем пользователям. Кроме того, проверим, как можно послать почту самому себе.

```
польз.  →  mail Patricia darrin roy dick⟨r⟩
UNIX    →          нет ответа (ожидает сообщения)
польз.  →  This is to let you know that I received⟨r⟩
        →  your message and will follow up on it. ⟨r⟩
        →          нажмите клавишу ctrl/d для отправки
        →          сообщения
UNIX    →  $
```

Указанное сообщение было послано четырем людям, имена которых перечислены в команде mail. При входе в систему или по специальному запросу почты они будут уведомлены о том, что для них есть почта. Можно послать несколько сообщений в разное время. Система снабдит их сведениями о времени отправки, имени отправителя и присоединит к ранее посланным. При запросе почты пользователь получит все сообщения.

Итак, вы послали почту самому себе. Ее можно получить простой командой mail. Любая почта, пришедшая после вашего входа в систему, может быть просмотрена командой mail. Момент принятия почты не сопровождается каким-либо уведомлением, поэтому вы сами должны просматривать почтовый ящик. Если вы забыли это сделать, почта останется в ящике до следующего вашего входа в систему.

Из примера видно, что система спрашивает, сохранить почту или нет. Для ответа «да» следует нажать клавишу «y» и «возврат каретки». Любой другой ответ приведет к удалению данного сообщения.

Узнать, если ли для вас почта, вы можете после входа в систему в любое время с помощью команды mail. Если почта есть, система напечатает ее, если нет — ответит сообщением «no mail» (почты нет).

```
польз.  →  mail⟨r⟩
UNIX    →  From dick Sat Jan 10 11:21:23 1981
        →  This is to let you know that I received
        →  your message and will follow up on it.
        →  save?
польз.  →  y⟨r⟩  «y» означает сохранить, «n» — удалить почту
UNIX    →  $
```

Выводы.

Mail — очень полезная команда, особенно если у вас нет возможности встретиться с адресатом, но вы хотите проинформировать его о чем-либо.

Необходимо помнить, что при отправке почты следует давать правильные имена пользователей. В противном случае почта может попасть к другому адресату или вообще потеряться.

7.1.2. СООБЩЕНИЕ ВСЕМ ПОЛЬЗОВАТЕЛЯМ

Команда: wall
Синтаксис: wall*
Действие: эта команда обычно используется администратором системы, когда нужно немедленно проинформировать всех пользователей о каком-либо важном событии. Таким событием может быть переполнение диска, ошибка устройства (магнитной ленты, АЦПУ и т. д.) или выключение машины.
Флаги: нет.

Пример.

Послать всем пользователям сообщение о том, что устройство печати неисправно и будет недоступно до следующего специального сообщения.

```
г.ольз. →  wall<r>
        →  The computer will be down for about one
        →  hour. Please logoff.
        →                нажмите ctrl/d для отправки сообщения
UNIX    →  Broadcast Message . . .  (сообщения системы...)
        →  The computer will be down for about one
        →  hour. Please logoff.
```

Выводы.

Как было указано ранее, команда wall удобна для немедленного информирования всех пользователей. Сообщение посылается только тем пользователям, которые в данный момент работают в системе.

7.1.3. СООБЩЕНИЕ ДРУГОМУ ПОЛЬЗОВАТЕЛЮ

Команда: write
Синтаксис: write имя [терминал]
Действие: команда позволяет посылать сообщение другому пользователю, работающему в настоящее время в системе. Команда write дает вам возможность беседовать друг с другом через свои терминалы. При этом у вас должен быть протокол связи, чтобы посылаемые туда и обратно сообщения не пересекались, как это обычно делается на радио или в других одноканальных системах связи. Если вы не желаете, чтобы вас прерывали сообщениями во время выполнения некоторой работы, то можете отменить их при помощи команды mesg.
Флаги: нет.

---

* Команда (выполняемый файл) wall находится в каталоге /etc и обычно вызывается полным именем: /etc/wall. — *Примеч. пер.*

Пример.

Вы вошлн в систему под нменем dick и хотите послать сообщенне пользователю с нменем darrin.

На термннале пользователя dick:

```
польз. →  write darrin<r>
UNIX  →                    ответа нет, система ждет ввода сообщения
```

На термннале пользователя darrin:

```
UNIX  →  message from dick tty1      (сообщение от dick tty1)
```

Darrin теперь может лнбо ждать сообщення от dick, либо дать команду «write dick». Следует быть осторожнымн, чтобы не послать сообщение в то время, когда другой пользователь сообщает вам нечто. Во нзбежанне такнх снтуаций нужен протокол связи. Простейшнй протокол состоит в том, чтобы каждый пользователь всегда знал, ждет лн его собеседннк ответа или готовнтся послать сообщение.

Рассмотрнм простой пример протокола связн:

по полученин сообщения «message from — tty?» ждите до тех пор, пока не прндет первое сообщенне с однночной буквой «о» в конце строки. Это означает, что пришла ваша очередь отвечать;

как только вам понадобнтся ответ, дайте отдельную букву «о» в конце строкн сообщення;

еслн вы закончилн разговор, дайте две буквы «оо».

Какой протокол вы установнте между собой — неважно; главное понять, как с ннм работать.

После того как вы дали команду write, можете начннать вводнть сообщенне. Каждым нажатием клавишн возврата каретки <r> вы будете посылать очередную строку другому пользователю.

Dick сообщает (после ввода команды write):

```
польз. →  This is to let you know that I am <r>
       →  working on project x and will be done <r>
       →  with the specification by tomorrow. oo <r>
.      →
.      →
.      →
```

На терминале пользователя darrin появится:

```
UNIX  →  This is to let you know that I am
      →  working on project x and will be done
      →  with the specification by tomorrow. oo
      →  EOF
      →  $
```

Как внднo нз прнмера, darrin получает точно такне же данные у себя на термннале, какне вводнт со своего термннала dick. Хотя это в примере н не показано, но, если бы darrin начал вводить сообщенне в то время, когда это же делал dick, пронзошло бы наложение вводимого текста на выводнмый на термннале у пользователя darrin.

Выводы.

Команда write в отличие от команды mail позволяет послать сообщение работающему в системе пользователю. И он получит его немедленно без выполнения отдельной команды mail. В некоторых случаях, если сообщение длинное, имеет смысл воспользоваться почтой, оповестив при этом пользователя командой write о факте отправки почты. Помните также о необходимости установления своего рода протокола связи при обмене сообщениями.

7.1.4. РАЗРЕШЕНИЕ ИЛИ ОТМЕНА СООБЩЕНИЙ

Команда: mesg
Синтаксис: mesg[n][y]
Действие: эта команда устанавливает или отменяет разрешение другим пользователям посылать вам сообщения. В некоторых случаях необходимо запретить пользователям вмешиваться своими сообщениями в вашу работу за терминалом.
Флаги: имеется два флага, управляющих принятием сообщений от других пользователей. Флаг «n» отменяет сообщения, идущие на ваш терминал, до тех пор, пока вы заново не войдете в систему или не дадите команду mesg с флагом «y». Флаг «y» устанавливает разрешение принятия сообщений от других пользователей. По умолчанию при входе в систему устанавливается разрешение приема сообщений.

Пример.

Предположим, что вы готовитесь начать редактирование файла и не хотите получать ни от кого сообщений на экране до тех пор, пока не завершите работу. Для этого необходимо сделать следующее:

```
польз. →  mesg n⟨r⟩
UNIX  →  $
```

Теперь никто не сможет послать вам сообщение, а если кто-то попытается, система уведомит его, что отправка сообщения запрещена.

По окончании редактирования своего файла вам следует дать команду mesg с установкой разрешения на прием сообщений для того, чтобы не потерять необходимую вам информацию.

```
польз. →  mesg y⟨r⟩
UNIX  →  $
```

С этого момента вы снова будете получать сообщения.

Выводы.

Следует иметь в виду, что командой mesg нужно пользоваться осторожно. Помните, что отменять сообщения не всегда в ваших

интересах, поэтому при первой возможности старайтесь восстанавливать разрешение на прием сообщений. Не забывайте также, что после того, как вы выйдете из системы и снова в нее войдете, вступит в силу умолчание, разрешающее прием сообщений.

7.1.5. ВОПРОСЫ

1. Можно ли послать почту сразу нескольким пользователям?
2. Могут ли несколько человек послать вам почту, если вы не работаете в системе?
3. Когда система UNIX сообщает вам о получении почты?
4. Есть ли в UNIX средство автоматического извещения всех пользователей, работающих в данный момент за терминалами, о некоторой неисправности, возникшей в системе?
5. Когда посылается и когда принимается сообщение, выдаваемое по команде write?
6. Кому вы можете послать сообщение?

7.2. КОМАНДЫ ОБРАБОТКИ ФАЙЛОВ

Необходимость выполнения небольших, но важных функций обработки файлов не вызывает сомнений. Однако без надлежащих средств это весьма трудная, если вообще разрешимая задача. UNIX — одна из систем, имеющих в своем распоряжении богатый набор средств, существенно «облегчающих жизнь» пользователю.

Описываемые здесь команды используются для сбора необходимой информации о файлах и каталогах.

7.2.1. ПОИСК ОДИНАКОВЫХ ИЛИ РАЗЛИЧНЫХ СТРОК ДВУХ ФАЙЛОВ

Команда: comm

Синтаксис: comm [—[123]] файл-1 файл-2

Действие: эта команда собирает информацию об одинаковых или разных строках в двух отсортированных файлах. Команда выводит в зависимости от заданных файлов от одной до трех колонок информации:

1) строки, встречающиеся только в файле1;
2) строки, встречающиеся только в файле2;
3) одинаковые строки в обоих файлах.

Флаги: флаги «123» обозначают номера колонок. Каждый заданный флаг означает, что соответствующая колонка не должна печататься. Так, флаг «—1» означает, что не будут напечатаны уникальные строки файла1. Флаг «—12» означает, что не будут напечатаны уникальные строки обоих файлов, т. е. будут напечатаны только одинаковые строки.

Примеры.

(1) Простейший способ нспользовання данной команды состонт в сравненнн двух отсортнрованных файлов без задания флагов. Предположим, что у нас есть два следующнх файла sfile1 н sfile2:

```
sfile1              sfile2
1 line one          1 line one
2 line two          2 line two
3 line three        3 line thre
4 line four         4 line four
5 line five         5 line five
6 line six          6 line six
7 line seven        7 line seven
```

Мы можем дать команду

```
польз.  →  comm sfile1 sfile2 ⟨r⟩
UNIX    →                 1 line one
        →                 2 line two
        →  3 line three
        →            3 line thre
        →                 4 line four
        →                 5 line five
        →                 6 line six
        →                 7 line seven
        →  $
```

Из результата сравнення мы вндим, что вывод состонт нз трех колонок:

колонка 1 содержнт строкн файла sfile1, отсутствующне в sfile2;

колонка 2 содержнт строкн файла sfile2, отсутствующне в sfile1;

колонка 3 содержит строки, которые нмеются в обоих файлах.

(2) Теперь воспользуемся флагами для вывода только необходнмых нам колонок. Зададнм однн из флагов для подавления вывода ненужной колонкн, напрнмер:

```
польз.  →  comm —1 sfile1 sfile2 ⟨r⟩
UNIX    →                 1 line one
        →                 2 line two
        →  3 line thre
        →                 4 line four
        →                 5 line five
        →                 6 line six
        →                 7 line seven
        →  $
```

Мы вндим, что уникальные строкн нз sfile1 не напечатались. Еслн воспользоваться флагом «—2» или «—3», то не напечатаются другне колонки.

(3) Мы можем комбиннровать флаги для печатн только нужных колонок. В предыдущем примере показано, как подавить вывод одной из колонок. Еслн нужно подавить вывод двух колонок,

задаются два флага. Например, для печати строк, встречающихся только в файле sfile1, нужно дать команду

```
польз. →  comm —23 sfile1 sfile2 ⟨r⟩
UNIX  →  3 line three
      →  $
```

Мы, конечно, можем с помощью флагов «—123» подавить вывод всех трех колонок, но такая команда не будет иметь никакого смысла.

Выводы.

Синтаксис команды comm похож на другие, поэтому мы можем, как и в других командах, направить вывод в некоторый файл или на вход некоторой команды.

7.2.2. ПРЕОБРАЗОВАНИЕ ФАЙЛА

Команда: dd

Синтаксис: dd [аргументы]...

Действие: эта команда позволяет задавать входной и выходной файлы и указывать различные виды преобразования информации между ними. Чаще всего она используется для чтения магнитных лент и записи на них с различным форматом данных (кодировкой, регистром букв, коэффициентом блокирования и т. п.).

Аргументы: возможны различные комбинации следующих аргументов:

if=имя	имя входного файла;
of=имя	имя выходного файла;
ibs=n	размер входного блока в байтах (512 по умолчанию);
obs=n	размер выходного блока в байтах (512 по умолчанию);
bs=n	размер входного и выходного блоков;
cbs=n	размер буфера преобразования в байтах;
skip=n	перед копированием пропустить n входных записей;
files=n	скопировать n файлов с входной ленты;
seek=n	установить выходной файл на запись с номером n перед началом копирования;
count=n	скопировать n входных записей.

Следующий аргумент составляется из нескольких подаргументов и используется при передаче данных из одной ЭВМ в другую.

conv=

ascii	преобразовать код EBCDIC в код ASCII;
ebcdic	преобразовать код ASCII в код EBCDIC;
ibm	несколько отличающееся преобразование кода ASCII в код EBCDIC;
lcase	преобразовать все буквы в строчные;
ucase	преобразовать все буквы в прописные;
swab	поменять местами байты в каждой паре байтов;
noerror	продолжать обработку в случае ошибки;

sync дополнять каждую входную запись до размера, заданного в ibs;

«,» выполнить несколько заданных преобразований.

Примеры.

(1) Простой случай вывода и ввода сблокированных данных. Вывести данные по 16 блоков в записи:

```
польз. →  dd of=/dev/rmt0 bs=16b . . . . .
```

Ввести данные, сблокированные по 16 блоков в записи:

```
польз. →  dd if=/dev/rmt0 bs=16b. . . . .
```

В обоих случаях используется прозрачный обмен с байторитентированным специальным файлом*, где /dev/rmt0 — имя устройства с магнитной лентой. Размер блока очевиден (512 байтов), однако следует помнить, что буква «b» — это размер записи в блоках.

(2) Попробуем теперь преобразовать все буквы файла в прописные при помощи аргумента conv=ucase. Пусть у нас есть файл file1. Его нужно преобразовать. Текущее состояние file1 таково:

```
1 line one
3 line three
7 line seven
6 line six
2 line two
5 line five
4 line four
```

Для перевода в прописные буквы дадим команду

```
польз. →  dd if=file1 conv=ucase <r>
UNIX   →  1 LINE ONE
       →  3 LINE THREE
       →  7 LINE SEVEN
       →  6 LINE SIX
       →  2 LINE TWO
       →  5 LINE FIVE
       →  4 LINE FOUR
       →  $
```

К верхнему регистру будут преобразованы только буквы. Мы могли бы задать аргумент «of=ab», и тогда вместо стандартного файла вывода результат был бы записан в файл ab.

(3) Другой полезный аргумент — «conv=swab». В ЭВМ PDP-11 байты в слове хранятся в обратном порядке. Собственно, для пользователя PDP-11 — это не имеет значения, но при переносе данных на другую 16-битовую ЭВМ, не меняющую порядок байтов в слове, могут возникнуть проблемы. Пусть перед переносом данных из одной ЭВМ в другую файл с обратным порядком байтов выглядел так:

---

* Здесь автор оперирует терминами, введенными в гл. 9. Кроме того, в примере реализуется возможность, которой нет в описании команды: если за размером записи следует буква b, это означает размер не в байтах, а в блоках по 512 байтов. — *Примеч. пер.*

1ILENO EN3
L NI EHTER
E 7ILENS VENE6
L NI EIS
X 2ILENT OW5
L NI EIFEV4
L NI EOFUR

Для преобразования его к прямому порядку следует дать команду

```
польз. →  dd if=ba conv=swab <r>
UNIX   →  1 LINE ONE
       →  3 LINE THREE
       →  7 LINE SEVEN
       →  6 LINE SIX
       →  2 LINE TWO
       →  5 LINE FIVE
       →  4 LINE FOUR
```

Выводы.

Как видно из примеров, команда dd удобна при передаче данных из машины в машину.

7.2.3. ВЫЯВЛЕНИЕ РАЗЛИЧИЙ МЕЖДУ ДВУМЯ ФАЙЛАМИ

Команда: diff
Синтаксис: diff[—efbh] файл-1 файл-2
Действие: эта команда определяет изменения в двух файлах, которые нужно сделать для приведения их в соответствие один другому. Такое средство позволяет запоминать версии файлов без необходимости хранения полных копий каждой версии файла.

Флаги: четыре флага дают дополнительные возможности:

—e наиболее важен, поскольку позволяет получить команды для редактора ed, с помощью которых первый файл можно привести к виду второго;

—b позволяет игнорировать все пробелы и символы табуляции в конце строки, а также считать эквивалентными любые последовательности таких символов;

—f вырабатывает список изменений, подобный списку флага —e, но несовместимый с редактором ed;

—h позволяет быстро обнаружить различия, но не всегда точно.

Примеры.

(1) Самый простой способ использования команды diff состоит просто в обнаружении различий файлов. Сделаем это для file1 и file2.

file1	file2
1 line one	1 line one
3 line three	3 line three
7 line seven	6 line six
6 line six	2 line two
2 line two	7 line seven

```
5 line five          5 line five
4 line four          4 line four
```

Выполним команду

```
польз.  →  diff file1 file2 ⟨r⟩
UNIX    →  3d2
        →  < 7 line seven
        →  5a5
        →  > 7 line seven
        →  $
```

Удалив строку с цифрой 7 (третья строка) и добавив такую же новую строку после пятой строки, мы сделаем file1 похожим иа file2. Хотя, иа первый взгляд, оба файла выглядят иеодииаково, фактически они оказались весьма близки один к другому. (2) В некоторых случаях хорошо иметь возможиость автоматически восстанавливать старое содержимое файла из иового. Это можно сделать при помощи флага —e. Он выявляет различия в файлах и вырабатывает результат, пригодный для использования редактором ed. Продемоистрируем этот результат иа файлах из предыдущего примера.

```
польз.  →  diff —e file1 file2 ⟨r⟩
UNIX    →  5a
        →  7 line seven
        →  .
        →  3d
        →  $
```

Этот результат теперь можно пропустить через редактор и привести file1 к виду file2.

Выводы.

Diff — очеиь удобиое средство в том случае, когда иужио восстаиовить старую копию файла, иапример, программы или документации. При создаии новой копии файла требуется запомнить только иебольшой файл различий, получеиный комаидой diff. Обычио ои бывает иамного меиьше самого файла.

7.2.4. ВЫЯВЛЕНИЕ РАЗЛИЧИЙ МЕЖДУ ТРЕМЯ ВЕРСИЯМИ ФАЙЛА

Команда: diff3

Синтаксис: diff3[—ex3] файл-1 файл-2 файл-3

Действие: команда сравнивает три версии файла и выявляет различия между ними. Осиовная иифорлация, выдаваемая командой, помечается следующими строками:

```
= = = =      все три файла различаются;
= = = = 1    отличается файл-1;
= = = = 2    отличается файл-2;
= = = = 3    отличается файл-3.
```

Выдаиной иифорлации достаточно для приведеиия старой версии файла к иовой.

Флаги: допустимы три флага:

—e вырабатывается результат, воспринимаемый редактором ed;
—x печатаются различия между всеми тремя файлами;
—3 печатаются различия между файлом-3 и двумя другими файлами.

Примеры.

(1) Первый пример показывает использование команды без флагов. Предположим, у нас есть три файла (file1, file2 и file3).

file1	file2	file3
1 line one	1 line one	1 line one
3 line three	3 line three	3 line three
7 line seven	6 line six	7 line seven
6 line six	2 line two	6 line six
2 line two	7 line seven	5 line five
5 line five	5 line five	2 line two
4 line four	4 line four	4 line four

Даем следующую команду:

```
польз.  →  diff3 file1 file2 file3 <r>
UNIX    →  ====2
        →  1 : 2a
        →  2 : 3,4c
        →  6 line six
        →  2 line two
        →  3 : 2a
        →  =====
        →  1 : 4,5c
        →  6 line six
        →  2 line two
        →  2 : 5a
        →  3 : 4c
        →  6 line six
        →...====3
        →  1 : 6a
        →  2 : 6a
        →  3 : 6c
        →  2 line two
        →  $
```

Команда снабжает нас информацией о различиях в каждом файле. Формат выдачи следующий:

1) файл: номер-строки a (добавить)
   Первую строку выдачи «1 : 2a» можно перевести так:
   файл-1 : строка 2 добавить,

что означает добавление некоторой информации после второй строки файла file1, с тем, чтобы привести ее к виду строки файла file2;

2) файл: номер-строки, номер-строки c (заменить)
   Вторую запись выдачи, состоящую из трех строк:
   2 : 3, 4c
   6 line six
   2 line two

можно перевести так:
   файл 2 : строка 3, строка 4 заменить,

что означает замену строк 3 и 4 файла file2 для того, чтобы привестн его к виду файла file1.

Оба эти формата описывают различия между тремя файлами. Строка «= = = =» обозначает номер отличающегося файла. (2) Здесь нспользуют те же три файла, что и в (1), но с флагом —e в команде diff3, чтобы получить результат, пригодный для редактора ed.

```
польз. →  diff3 —e file1 file2 file3 <r>
UNIX →  6a
     →  2 line two
     →  .
     →  4,5c
     →  6 line six
     →  .
     →  $
```

В этом пнмере показаны изменення (в формате редактора ed), которые необходнмо сделать в файле-1 для прнведення его к внду файла-2 н файла-3. Для преобразования файла-1 только к файлу-3 следует прнменить флаг «—3».

Выводы.

Команда diff3 обычно работает с тремя верснямн одного файла. Она предоставляет средства хранення изменений в трех разлнчных версиях файла и позволяет восстанавлнвать каждую нз версий по любой другой.

7.2.5. ПОИСК СТРОК С ЗАДАННЫМ ШАБЛОНОМ

Команда: grep

Синтаксис: grep [флаг]... выраженне [файл]

Действие: нспользуется, когда требуется найти строкн с заданным шаблоном в одном илн более файлах. Для одного файла нскомый шаблон можно обнаружить и с помощью редактора ed, команда же grep обычно применяется для поиска в нескольких файлах. Она обнаружнвает все вхождення шаблона во всех заданных файлах. Шаблоном в команде grep может быть любое регулярное выражение, определяемое правилами редактора ed.

Флаги: могут быть заданы следующие флаги:

—v печатаются все строки, не содержащне шаблона;
—c печатается только число обнаруженных строк с шаблоном;
—l перечисляются имена файлов, содержащих искомые строкн;
—n перед каждой обнаруженной строкой печатается ее порядковый номер в файле;
—b перед каждой обнаруженной строкой печатается номер блока, в котором она содержится;
—s ничего не выводится, вырабатывается только статус завершення команды;
—h не печатаются нмена файлов перед строками;

—y строчные буквы в шаблоне считаются совпадающими как со строчными, так и с прописными буквами в файлах;
—e используется перед шаблоном, который начинается с символа «—».

Примеры.

(1) Мы описывали предыдущие команды с помощью нескольких файлов file1, file2 и file3, похожих один на другой.

file1	file2	file3
1 line one	1 line one	1 line one
3 line three	3 line three	3 line three
7 line seven	6 line six	7 line seven
6 line six	2 line two	6 line six
2 line two	7 line seven	5 line five
5 line five	5 line five	2 line two
4 line four	4 line four	4 line four

Для обнаружения всех вхождений строки «seven» следует дать команду

```
польз. →  grep seven file1 file2 file3 ⟨r⟩
UNIX  →  file1 : 7 line seven
      →  file2 : 7 line seven
      →  file3 : 7 line seven
      →  $
```

Мы получили информацию, содержащую имя файла и распечатку всей строки с заданным шаблоном. Ту же команду можно было дать двумя другими способами и получить такую же информацию:

1) grep seven file? ⟨r⟩
2) grep seven fil* ⟨r⟩

В обоих случаях мы должны быть уверены, что шаблон имени файла обозначает только наши три файла (т. е. что нет файлов типа file5 или filtemp).

(2) В предыдущем примере мы обнаружили строки, содержащие шаблон, но без указания на их местоположение в каждом файле. Для получения номера строки каждого вхождения шаблона следует задать флаг —n. Например:

```
польз. →  grep —n seven file? ⟨r⟩
UNIX  →  file1 : 3 : 7 line seven
      →  file2 : 5 : 7 line seven
      →  file3 : 3 : 7 line seven
      →  $
```

Единственное отличие второго примера от первого в том, что в него включена печать номеров строк. Каждая строка выдачи содержит (слева направо): имя файла, номер строки, собственно строку.

(3) Мы ищем шаблон, состоящий из строчных и прописных букв. Ввиду большого числа комбинаций было бы крайне затруднительно отдельно задавать все возможные шаблоны. Флаг —y поз-

воляет нам единственным шаблоном обозначить все такие возможности.

Предположим, мы собираемся редактировать файл, где есть некоторые прописные буквы. Файл filex содержит следующие данные:

```
filex
1 line one
3 line three
7 line seven
6 line six
2 line Two
5 line Five
4 line Four
```

Мы дали команду

```
польз.  →  grep —y f filex ⟨r⟩
UNIX    →  5 line Five
        →  4 line Four
        →  $
```

В данном случае отыскивается все, что содержит «f» или «F». (4) Мы можем также комбинировать флаги, если это необходимо. В предыдущем примере требовалось найти только прописные буквы «F». Если бы мы захотели печатать также номера строк (флаг — n), то могли бы дать такую команду:

```
польз.  →  grep —y —n f filex ⟨r⟩
UNIX    →  6 : 5 line Five
        →  7 : 4 line Four
        →  $
```

Заметим, что в предыдущих двух примерах имя файла на печать не выводилось. Так бывает всегда, когда задается только один файл. В последнем случае выводились номер строки и сама строка.

Выводы.

Команда grep может быть полезной при просмотре нескольких файлов. Для примера возьмите какие-нибудь программы на языке Си и попробуйте найти некоторый шаблон (например, имя переменной с целью ее изменения или удаления).

7.2.6. ВОСЬМЕРИЧНЫЙ ДАМП

Команда: od

Синтаксис: od[—bcdox] файл [[+] смещение [.][b]].

Действие: программа od предоставляет средства выдачи дампа файла или его части в одном или нескольких форматах, как это задано флагами команды.

Флаги: существует несколько флагов, дающих возможность распечатать содержимое файла в различных форматах:

b каждый байт файла интерпретируется как восьмеричное число;

c каждый байт файла интерпретируется как символ в коде ASCII, причем неграфические символы выводятся в следующем специальном формате:
1) нулевой байт = \0
2) возврат на шаг = \b
3) перевод формата = \f
4) перевод строки = \n
5) возврат каретки = \r
6) горизонтальная табуляция = \t
7) остальные = \ddd
где ddd — трехцифровое восьмеричное число;

d каждое слово интерпретируется как десятичное число;

o каждое слово интерпретируется как восьмеричное число (данный флаг принят по умолчанию);

x каждое слово интерпретируется как шестнадцатеричное число.

Примеры.

(1) Возьмем простой восьмеричный дамп файла из двух строк. Содержимое файла filea равно:

```
filea
1 1 2 1 1
2 2 1 2 2
польз. →   od filea <r>
UNIX →
     →   0000000 030440 030440 031040 030440
     →   0000000 030412 031040 031040 030440
     →   0000020 031040 031012
     →   0000024
     →   $
```

Это простой восьмеричный дамп двухстрочного файла filea.

(2) Здесь рассмотрим тот же самый восьмеричный дамп, однако флаг — b выводит не слова, а отдельные байты.

```
польз. →   od —b filea <r>
UNIX →
     →   0000000 061 040 061 040 062 040 061 040
     →   0000000 061 012 062 040 062 040 061 040
     →   0000020 062 040 062 012
     →   0000024
     →   $
```

В примере (2) (в отличие от примера (1)) каждый символ файла filea представлен отдельно.

(3) Чаще всего для представления текста в коде ASCII с распечаткой неграфических символов в команде od используется флаг —c. Во многих случаях программа работает неправильно из-за того, что в тексте встречается неграфический символ:

```
польз. →   od —c filea <r>
UNIX →
     →   0000000 1 1 2 1 1 \n 2 2 1
     →   0000020 2 2\n
     →   0000024
     →   $
```

В данном примере мы видим два неграфических символа перевода строки «\n». Если не предполагать их появление в этих местах файла, то без команды od их трудно обнаружить в тексте.

Выводы.

Основное назначение команды od — отладка, как программ, так и документов, предназначенных для форматирования.

7.2.7. ПОСТРОЕНИЕ ТАБЛИЦЫ С ОГЛАВЛЕНИЕМ БИБЛИОТЕКИ

Команда: ranlib
Синтаксис: ranlib библиотека
Действие: эта команда используется для обработки библиотеки, содержащей объектные модули программ, которые затем будут объединяться редактором связей. Ranlib помещает в начало библиотеки таблицу с оглавлением, позволяющую редактору связей за один проход по библиотеке отыскивать все модули, требуемые для компонуемой программы. Если такой таблицы нет, то модули в библиотеке должны быть выстроены в порядке, позволяющем редактору связей применить однопроходный поиск по библиотеке*.
Флаги: нет.

Пример.

Для выполнения команды нужно ввести только ее имя и имя библиотечного файла. Будет построена таблица с оглавлением. Любые изменения, происходящие в библиотеке после ее построения, приводят к тому, что таблица уже не соответствует текущему содержимому библиотеки. Для восстановления соответствия нужно еще раз выполнить команду ranlib. Если вы забудете это сделать, то при обращении к библиотеке редактор связей сообщит об устарении таблицы.

```
польз. →  ranlib library <r>
UNIX →  $
```

При нормальном завершении команда не выводит никаких сообщений. Будет построена таблица с оглавлением библиотеки (имя таблицы —«__.SYMDEF») и вставлена в начало библиотечного файла.

Выводы.

Команда ranlib используется только в связи с редактором ld. При этом необходимость реорганизации библиотеки, требуемой редактором связей, отпадает.

---

* Команда ranlib не входнт в каноннческую, 7-ю, версню UNIX, а редактор связей ld в каноннческой версни не умеет работать с таблицей оглавленя, построенной ranlib. — *Примеч. ред.*

7.2.8. ПОДСЧЕТ ЧИСЛА СЛОВ

Команда: wc
Синтаксис: wc[—lwc] [файл...]
Действие: эта команда выдает количество строк, слов и символов в одном или более файлов. Строки в файле разделяются символом перевода строки «\n», слова — пробелами, горизонтальной табуляцией или переводом строки. Символом считается любой символ файла.
Флаги: флаги позволяют задать выдачу только требуемой статистики:

l подсчет числа строк в файле,
w подсчет числа слов в файле,
c подсчет числа символов в файле.

Примеры.

(1) Давайте сначала применим одиночную команду к простому файлу. Содержимое файла:

```
польз.  →  cat file1
UNIX    →  1 line one
        →  3 line three
        →  7 line seven
        →  6 line six
        →  2 line two
        →  5 line five
        →  4 line four
        →  $
```

Теперь можно выполнить команду

```
польз.  →  wc file1 <r>
UNIX    →  7   21   83 file1
        →  $
```

Вывод команды состоит из числа строк (7), числа слов (21), числа символов (83) и, наконец, имени файла.
(2) Если нужно вывести не все данные, пользуются флагом. Например, если мы хотим узнать только число строк файла, то даем следующую команду.

```
польз.  →  wc —l file1 <r>
UNIX    →  7 file1
        →  $
```

Вывод содержит только число строк и имя файла.
(3) Команда wc может выдать статистику для большого количества файлов. Например, мы желаем узнать, много ли строк и символов содержится в группе программ, используемых в некоторой системе. Предположим, что этой группе программ принадлежат файлы, имена которых начинаются с fil. Для подсчета общего числа строк и символов выполним такую команду:

```
польз.  →  wc —lc fil* <r>
UNIX    →  7  83 file1
        →  7  83 file1.c
```

```
→   1    5 file1.o
→   7   83 file2
→   7   83 file3
→   0   83 filen
→  29  420 total
→  $
```

В каждой строке вывода слева направо печатается число строк, число символов и имя файла. Последняя строка содержит общее число (сумму) строк и снмволов.

Выводы.

Команда wc нужна всякий раз, когда мы хотим узнать число строк, слов и символов одного или нескольких файлов.

7.2.9. ВЫВОД ОДИНАКОВЫХ СТРОК ФАЙЛА

Команда: uniq

Синтаксис: uniq [-флаги[+n][−n]][вход][выход].

Действие: эта команда выявляет одинаковые соседние строки файла. По умолчанию (в команде без флагов) все соседние одинаковые строки, кроме одной, удаляются. Например, если обнаружено 4 идущих подряд одинаковых строки, то строки 2, 3 и 4 будут удалены. Выходной файл будет состоять только из разных строк.

Флаги: возможны три флага:

u выводятся только неодинаковые строки;

d выводятся только одинаковые строки;

c одинаковые строки удаляются, как и в случае отсутствия флагов, но в начало каждой строкн помещается число ее вхождений в файл. Флаги +n и —n, где n — целое положительное число, задают пропуск информации в начале каждой строкн перед выполненнем сравнения:

—n первые n полей пропускаются перед сравнением. Поле определяется как строка, не содержащая символов пробела и табуляции и отделяемая от других строк такими символами;

+n первые n символов поля перед сравнением пропускаются. Сначала пропускаются поля, затем символы.

Примеры.

(1) Начнем с простого файла и построим новый файл, удалив все вторые и последующие копин одинаковых строк. Файл f1 содержит следующие данные:

```
1 line one
3 line three
3 line three
7 line seven
6 line six
2 line two
2 line two
```

```
5 line five
4 line four

польз.  →   uniq f1 f2 <r>
UNIX    →   $
польз.  →   cat f2 <r>
UNIX    →   1 line one
        →   3 line three
        →   7 line seven
        →   6 line six
        →   2 line two
        →   5 line five
        →   4 line four
        →   $
```

Как видно из примера, новый файл f2 содержит только неповторяющиеся строки. Если существует более двух вхождений строки, то эффект будет такой же, т. е. все копии, начиная со второй, будут удалены. Необходимо отметить также, что файл должен быть упорядочен (см. команду sort), в противном случае будут удалены не все одинаковые строки.

(2) Воспользуемся флагом —u для вывода только неповторяющихся строк.

```
польз.  →   uniq —u f1<r>        вывод направлен на терминал
UNIX    →   1 line one
        →   7 line seven
        →   6 line six
        →   5 line five
        →   4 line four
        →   $
```

Две строки (2 и 3) имели копии и поэтому не появились на экране терминала.

(3) В некоторых случаях желательно вывести только повторяющиеся строки. Для этого существует флаг —d.

```
польз.  →   uniq —d f1<r>
UNIX    →   3 line three
        →   2 line two
        →   $
```

(4) Теперь подсчитаем с помощью флага —c число вхождений каждой строки.

```
польз.  →   uniq —c f1<r>
UNIX    →   1 1 line one
        →   2 3 line three
        →   1 7 line seven
        →   1 6 line six
        →   2 2 line two
        →   1 5 line five
        →   1 4 line four
        →   $
```

Первое число — это число вхождений данной строки. Таким образом, строки 2 и 5 имеют по два вхождения, в то время как остальные — по одному.

(5) В некоторых случаях желательно сравнивать только части строк. Флаги +n и —n предназначены именно для этой цели и

позволяют нам пропускать начальную часть каждой строки, а также заданное число соответственно строк и символов перед сравнением. Возьмем файл, содержащий строки с повторяющимся первым полем. Такой файл (по имени f3) может иметь следующий вид:

```
1 line one
3 line three
3 line three3
7 line seven
6 line six
2 line two
2 line two2
5 line five
4 line four
```

```
польз.  →   uniq —1 f3 <r>
UNIX    →   1 line one
        →   3 line three
        →   3 line three
        →   7 line seven
        →   6 line six
        →   2 line two
        →   2 line two2
        →   5 line five
        →   4 line four
        →   $
```

Легко видеть, что, поскольку у строк не совпадают только первые поля, а мы первое поле пропустили при сравнении, выводятся все строки (как несовпадающие). Если бы эта возможность не была использована, строки 3 и 7 не попали бы в вывод.

Выводы.

Помните, что повторяющиеся строки должны быть рядом, чтобы команда могла их обнаружить. Этого можно добиться, выполнив команду sort. В приведенных примерах мы не задавали выходной файл, поэтому вывод направлялся на терминал. Как правило, выходной файл в команде uniq задается.

7.2.10. РАЗБИЕНИЕ ФАЙЛА НА ЧАСТИ

Команда: split

Синтаксис: split[—n][файл[имя]]

Действие: эта команда разбивает файл на части по n строк (по умолчанию n=1000), в результате чего образуются выходные файлы в количестве, необходимом для представления входного файла. Если задано имя выходного файла (аргумент «имя»), то генерируется последовательность файлов с данным именем и буквами aa, ab, ac, ... в конце.
Если имя выходного файла не задано, используется имя «x».

Флаги: имеется единственный флаг —n, где n — число строк в выходных файлах.

Пример.

Воспользуемся файлом f1 (см. пример из раздела 7.2.9) (команда uniq). Разобъем этот файл на файлы по 2 строки в каждом с именами, начинающимися с ff.

```
польз.  →  split —2 f1 ff <r>
UNIX    →  $
польз.  →  ls * <r>
UNIX    →  f1
        →  ffaa
        →  ffab
        →  ffac
        →  ffad
        →  ffae
        →  $

польз.  →  cat ffaa ffab ffac ffad ffae <r>
UNIX    →  1 line one
        →  3 line three
        →  3 line three
        →  7 line seven
        →  6 line six
        →  2 line two
        →  2 line two
        →  5 line five
        →  4 line four
        →  $
```

Если бы мы распечатали файл f1, вырисовалась бы та же картина, что и в предыдущем примере, только теперь строки содержались бы в пяти различных файлах.

Выводы.

Основное назначение комаиды split — разбиение данных на небольшие файлы, когда: 1) имя стаиовится трудио управлять или 2) они выходят за граинцы, устанавливаемые некоторыми программами (такими, например, как редактор ed).

7.2.11. СОРТИРОВКА И СЛИЯНИЕ ФАЙЛОВ

Команда: sort

Синтаксис: sort [—флаги...][+поз1[—поз2]]... [—о имя] [— Т каталог][имя]...

Действие: эта комаида сортирует или соединяет файлы, помещая результат в заданный файл (по умолчанию в стандартный вывод). Если ключ сортировки не задан, в сравнении участвует вся строка. Можно задать сортировку по одному или нескольким ключам в указаином специальиыми флагами команды порядке.

Флаги: команда предоставляет следующие возможности, задаваемые флагами:

b при сравнении полей игнорируются пробелы и табуляции в иачале строки;

c проверяется, отсортирован ли входной файл в соответствии с заданными правилами; не выводится никакой информации, если только файл не отсортирован как требуется;

d «словарная сортировка»: в сравнении участвуют только буквы, цифры и пробелы;

f прописные буквы трактуются как строчные;

i при нечисловых сравнениях игнорируются символы (восьмеричные), не входящие в диапазон ASCII Ø40— Ø176;

m слияние файлов; предполагается, что входные файлы уже отсортированы;

n сортировка по арифметическому значению. Числовая строка может состоять из пробелов, знака минус, нуля или более цифр с десятичной точкой;

o имя, идущее за этим флагом, задает выходной файл. Если выходной файл не задан, подразумевается стандартный вывод;

r задается обратный порядок сортировки;

tx буква t указывает на то, что вместо принятого по умолчанию пробела в качестве разделителя полей будет использоваться горизонтальная табуляция;
Произвольный символ x будет разделителем, если он задан после t;

T задает имя каталога, где будут размещаться временные файлы сортировки;

u если одному ключу соответствует несколько строк, выводится только одна из них.

Спецификация ключей, по которым производится сортировка, выполняется с помощью флагов +поз1 и —поз2. Каждая пара таких флагов задает ключ сортировки, начинающийся с позиции «поз1» и заканчивающийся позицией «поз2». Флаги поз1 и поз2 имеют форму m.n, где m означает, что нужно пропустить m полей от начала строки, а u означает, что нужно пропустить n символов от начала заданного поля. Сначала пропускаются поля, затем символы. Если опущен флаг —поз2, то ключ располагается до конца строки.

Примеры.

(1) Сначала отсортируем файл по значению всей строки. Другими словами, ключом у нас будет полная строка. Такая сортировка выполняется по умолчанию, если не заданы никакие ключи. Файл file1, который мы будет сортировать, содержит следующие данные:

```
1 line one
3 line three
7 line seven
6 line six
2 line two
5 line five
4 line four
```

```
польз. →  sort file1 <r>
UNIX →  1 line one
     →  2 line two
     →  3 line three
     →  4 line four
     →  5 line five
     →  6 line six
     →  7 line seven
     →  $
```

(2) Теперь мы можем отсортировать тот же файл в обратном порядке, воспользовавшись флагом —r.

```
польз. →  sort —r file1 <r>
UNIX →  7 line seven
     →  6 line six
     →  5 line five
     →  4 line four
     →  3 line three
     →  2 line two
     →  1 line one
     →  $
```

В первых двух примерах ключом сортировки была вся строка. Если бы мы пожелали сортировать файлы только по числовому значению первого поля, следовало бы задать флаг —n. Если поле первое в строке, то флаг ключа не нужен. Однако если поле расположено где-то в другом месте строки, флаг ключа необходим. Рассмотрим примеры на сортировку по ключам.

(3) Сначала будем сортировать по первому полю (числовому).

```
польз. →  sort —n file1 —o outfile <r>
UNIX →  $
```

Выходной файл получается тот же, что и в примере (1), но в этом случае мы разместили результат в файле outfile.

(4) Если ключ сортировки расположен не в начале строки, мы можем воспользоваться числовым флагом для указания расположения ключа. Предположим, что ключ (числовой) располагается не в первом, а в последнем поле строки. Тогда мы можем дать следующую команду:

```
польз. →  sort —n +2 file2 —o outfile <r>
UNIX →  $
```

Результат будет помещен в файл с именем outfile. Ключ сортировки здесь был третьим полем в строке, поскольку +2 означает пропуск двух полей.

(5) В команде сортировки можно задавать сразу несколько флагов при условии, что они не противоречат один другому. Например, мы можем сортировать данные по арифметическому значению и заказать обратный порядок сортировки.

```
польз. →  sort —nr file1 —o file2 <r>
UNIX →  $
```

Результат отсортирован по первому (числовому) полю, причем наибольшее значение идет первым, наименьшее — последним.

Выводы.

Команда сортировкн представляет собой мощное средство, которое может применяться для решения многих задач. Примеры демонстрируют лишь небольшую долю возможностей команды, но на примерах можно научиться проверять более сложные варианты с помощью различных флагов.

7.2.12. ВОПРОСЫ

1. Вам требуется прочнтать на своей машине, нспользующей код ASCII, магнитную ленту драйвер магнитной ленты имеет нмя /dev/mt8, содержащую снмволы в коде EBCDIC. Какую команду н с какнми аргументамн следует применить?

2. Какая команда нужна для генерации последовательности команд редактора ed, устраняющнх различия между двумя файламн file1 н file2?

3. В вашем текущем каталоге шесть файлов с текстовой информацией. Какую команду нужно дать для обнаружения всех вхождений (во всех файлах) шаблона «Syntax»?

4. Что делают следующне команды:

а) wc file1
б) split —10 file1 F
в) sort —r file1 —o file2

7.3. УПРАВЛЕНИЕ ВЫПОЛНЕНИЕМ ПРОГРАММЫ

Представляемый здесь набор команд относится к управлению выполненнем программ. Эти команды особенно полезны в командных файлах ннтерпретатора shell (см. гл. 6), так как позволяют выполнять программы в указанные моменты времени*, ожндать завершення процесса н т. п.

7.3.1. ВЫВОД АРГУМЕНТОВ

Команда: echo
Синтаксис: echo[—п] [аргумент]...
Действие: эта команда выводит в стандартный файл вывода заданные ей аргументы, разделяя их пробелами н завершая вывод переводом строки. Она особенно полезна в командных файлах для выдачи сообщеннй о последовательно выполняемых командах.
Флаги: единственный флаг —п предоставляет возможность отменнть перевод строки после вывода аргументов.

---

* Вопросы выполнения команд в указанные моменты времени здесь не рассматриваются. — *Примеч. ред.*

Прнмеры.

(1) Требуется создать командный файл, выполняющнй несколько команд н печатающнй сообщенне о начале выполненя каждой команды.

Сначала мы создаем командный файл с нменем echo:

```
echo starting command1
command1
echo starting command2
command2
echo starting command3
command3
echo end of shell file
```

Теперь мы можем выполннть этот командный файл с помощью команды shell (см. гл. 8).

```
польз.  →   sh echo <r>
UNIX    →   starting command1
        →   starting command2
        →   starting command3
        →   end of shell file
        →   $
```

Команда echo удобна для слеженя за процессом выполнення командного файла нлн для проверкн правнльностн выполнення командного файла.

(2) Мы можем также выводнть н нмена аргументов, которые передаются командам нашего файла. Это удобно для проверкн того, что нужные аргументы передаются в нужное время, нлн просто для создания протокола выполнения командного файла. Создаднм командный файл echo1:

```
echo $1
command1 $1
echo $2
command2 $2
echo end of shell file
```

Этот файл можно вызвать по команде:

```
польз.  →   sh echo1 test1 test2 <r>
UNIX    →   test1
        →   test2
        →   end of shell file
        →   $
```

Как внднo нз данного прнмера, все что мы сделалн — напечаталн аргументы, переданные командному файлу. Во многнх случаях это не имеет смысла, но иногда мы будем вызывать командный файл нз другого командного файла и аргументы будут передаваться нз одного файла в другой неявно.

Выводы.

Команда echo нужна во многих случаях. Некоторые из ннх мы продемонстрировали. Но это не еднnственные возможные применення, так, напрнмер, здесь не показана выдача с помощью

команды echo сообщенй прн дналоге между пользователем и программой. После того как вы поблнже познакомнтесь с командой echo, вы оцените н другие ее возможностн.

7.3.2. УНИЧТОЖЕНИЕ ПРОЦЕССА

Команда: kill

Синтаксис: kill[—флаг] процесс...

Действие: эта команда используется для завершения процесса в том случае, еслн он мешает снстеме нли если вы решнлн, что он вам больше не нужен. Процесс задается чнсловым ндентнфнкатором, который можно узнать с помощью команды ps. Уннчтожнть процесс нмеет право лнбо тот, кто его ннницнровал, либо привнлегированный пользователь.

Флаги: еднностенный флаг, допустнмый в команде kill, — номер снгнала. С его помощью можно завершать процессы, нгнорирующне обычный снгнал. Например, флаг —9 вызывает безусловное завершение процесса. Вы можете полностью завершнть работу снстемы командой «kill —1 1»*.

Пример.

Предположнм, что вы задали отложенную печать файла. Поскольку прн этом вы немедленно получаете управленне, то для завершення процесса отложенной печати следует прежде всего узнать номер процесса (по команде ps). Как только вы его узналн, можете давать команду kill.

```
польз.  →   ps <r>
UNIX    →   PID   TTY    TIME        CMD
        →   1455  co     0:01
        →   3130  2      0:03
        →   4317  2      0:41
        →   4354  ?      <defunct>
        →   4355  2      0:00
        →   4356  2      0:03        lpr
        →   $
```

В данном случае вы запросилн ннформацию только о свонх актнвных процессах. Ваш терминал нмеет номер 2, и вы можете выбрать любой процесс для завершения. Поскольку вы пожелали завершнть процесс, выполняющнй отложенную печать (команду lpr), то определяете его ндентифнкатор — чнсло 4356.

```
польз.  →   kill 4356 <r>
UNIX    →   $
```

---

* Команда kill, вообще говоря, не уничтожает процесс, а посылает ему сигиал. Процесс может игнорировать или перехватывать сигнал. Если процесс этого ие предусмотрел, система отрабатывает стаидартиую реакцию иа сигнал, заключающуюся в уничтожеиии процесса. Сигиал с иомером 9 не может быть перехвачеи или проигнорирован и поэтому всегда влечет за собой уиичтожеиие процесса. — *Примеч. ред.*

Чтобы убедиться в правильном завершении процесса, можете дать еще одну команду ps. Процесса 4356 не должно быть в таблице активных процессов.

```
польз.  →   ps <r>
UNIX    →   PID   TTY   TIME          CMD
        →   1455  co    0:01
        →   3130  2     0:03
        →   4317  2     0:41
        →   4354  ?     <defunct>
        →   4355  2     0:00
        →   $
```

Выводы.

Командой kill следует пользоваться с осторожностью, поскольку вы можете уничтожить не тот процесс либо вообще завершить свою работу, и система выведет вас из числа текущих пользователей. Хорошим способом безусловного завершения процесса является использование флага —9.

7.3.3. ЗАДЕРЖКА ВЫПОЛНЕНИЯ КОМАНДЫ

Команда: sleep
Синтаксис: sleep время
Действие: sleep задерживает выполнение команды на заданный промежуток времени в секундах. Это дает возможность отложить выполнение некоторой команды до тех пор, пока не завершится какое-то другое действие.
Флаги: нет.

Пример.

Предположим, что мы хотим напомнить кому-то об обязанности выполнить определенную работу через час, считая от текущего момента времени. Если мы не собираемся быть здесь в это время, то можем воспользоваться командой

```
польз.  →   sleep 3600 <r>
UNIX    →   write sam <r>
        →   пожалуйста, выполните определенную задачу
```

Система будет ждать 3600 секунд перед тем, как послать сообщение. Трудность в том, что в течение этого времени вы не должны выходить из системы. В следующей главе показано, как создавать процедуры интерпретатора shell для выполнения их в более позднее время. Сейчас же нам важно понять работу команды sleep.

Выводы.

Существует много случаев, когда нужно выполнять эту команду. Позднее мы познакомимся со многими из них.

7.3.4. ПОНИЖЕНИЕ ПРИОРИТЕТА КОМАНДЫ

Команда: nice
Синтаксис: nice [—число]команда[аргументы]
Действие: эта команда позволяет выполнить другую команду с более низким приоритетом, т. е. без особого влияния на время работы других команд, выполняемых в это же самое время в системе. Аргумент «—число» обозначает, на сколько нужно уменьшить приоритет команды. Чем больше число, тем меньше, следовательно, будет приоритет. Наибольшее число, которое может быть задано, равно 20. Если число не задано, подразумевается 10.
Флаги: нет.

Пример.

Мы хотим распечатать содержимое каталога ленты, причем таким образом, чтобы это не повлияло на нашу оперативную работу по редактированию файла.

```
польз.  →  nice —20 tar c0 working& <r>
UNIX    →  234              идентификатор процесса
        →  $
польз.  →  ed file1 <r>
        →  .
        →  .
        →  .
```

Единственная разнница по сравнению с обычным запуском команды tar заключается в том, что здесь каталог будет печататься с более низким приоритетом и будет предоставляться больше времени центрального процессора для редактнрования нашего файла file1.

Выводы.

Придет время, когда для нас команда nice будет нметь большее значение, чем теперь (как и команда sleep). Чаще всего nice применяется администратором при нормальной и сильной загруженности системы.

7.3.5. ДУБЛИРОВАНИЕ СТАНДАРТНОГО ВЫВОДА

Команда: tee
Синтаксис: tee [флаг]...[файл]...
Действие: эта команда читает информацию из стандартного ввода н выводит ее одновременно на терминал (в стандартный вывод) и в заданные файлы. Чаще всего она используется при выполнении командных файлов (см. гл. 8).
Флаги: имеется два флага.
—i игнорировать прерывания;

—a вывод будет добавлен к файлу, в отличие от принятого по умолчанию создания нового файла.

Пример.

Необходимо выполнить ряд команд и запомнить порядок их выполнения. Пусть имеются следующие команды:

```
echo command1|tee —a file1
command1
echo command2|tee —a file2
command2
echo command3|tee —a file3
command3
```

При их выполнении получим:

```
UNIX  →  command1
      →  command2
      →  command3
      →  $
польз.→  cat file1 <r>
UNIX  →  command1
      →  command2
      →  command3
      →  $
```

Из примера мы видим, что данные, записанные в файл, совпадают с данными, выведенными на терминал. Tee удобно использовать при запуске ряда команд, когда мы не желаем следить за их выполнением, но хотим знать, что они завершились правильно.

Выводы.

О применении команды tee подробнее говорится в гл. 8 (о командных файлах).

7.3.6. ВОПРОСЫ

1. Какова цель команды echo?
2. Вы создали процесс, который стал ненужен. Его идентификатор 102. Какую команду нужно дать для завершения процесса? Как убедиться в том, что процесс завершен?
3. Как обеспечить выполнение долго работающей команды, минимально влияющей на выполнение любых других текущих команд?

7.4. ИНФОРМАЦИОННЫЕ КОМАНДЫ

Часто бывает необходимо получить информацию о состоянии системы или какую-либо другую. Очень полезно иметь простые и легкие в работе команды, предоставляющие информацию о файлах и каталогах, величине свободного места в файловой сис-

теме, количестве блоков, занимаемых на диске вашим каталогом, о работающих в системе, о времени и пр.

Такие команды снабжают пользователя постоянно требующимися сведениями. Рассмотрим эти команды.

7.4.1. ВЫВОД СОДЕРЖИМОГО КАТАЛОГА

Команда: ls

Синтаксис: ls [—флаги...]имя...

Действие: эта команда предоставляет информацию о файлах и каталогах, в частности о коде защиты, времени их последнего изменения и т. п. Команда ls без аргументов выводит в стандартный файл вывода отсортированный в алфавитном порядке список имен всех файлов и каталогов, содержащихся в текущем каталоге пользователя.

Флаги: есть несколько флагов для упорядочения и выбора различных элементов списка:

l выводит полную информацию о файле (см. гл. 4);

t сортирует список по времени модификации (последний измененный выводится первым);

a выводит все файлы каталога. По умолчанию имена, начинающиеся с «.», не выводятся;

s выводит размер файлов в блоках, включая косвенные блоки;

d выводит информацию только о каталогах (в формате, соответствующем флагу —1);

r сортирует список в обратном порядке;

u сортирует список по времени последнего доступа (последний, к которому был доступ, выводится первым);

c сортирует список по времени модификации индексного дескриптора (смена прав доступа и т. п.);

i выводит номер индексного дескриптора в первой колонке (для каждого файла);

f для каждого подкаталога выводит его содержимое. Этот флаг выключает флаги —l, —t, —s, —r и флаг —a. Файлы выводятся в том порядке, в каком они расположены в данном каталоге;

g вместо идентификатора владельца печатается идентификатор группы.

Примеры.

(1) Из гл. 4 мы уже знаем, как пользоваться командой ls. Теперь посмотрим, как можно применять различные флаги команды. Рассмотрим первый флаг, это флаг —a, предоставляющий возможность распечатать полное содержимое каталога (обычно система подавляет вывод файлов с именами, начинающимися с «.» и «..»).

```
польз. →   ls —al <r>
UNIX →
```

```
total 829
drwxrwxr-x 4   dick   1376   Jan 12 10:30
drwxrwxr-x 13  dick   400    Jan 17 10:49   ..
-rw-rw-r-- 1   dick   5705   Dec 21 15:35   ar
drwxrwxr-x 2   dick   704    Dec 20 13:48   book
-rw-rw-r-- 1   dick   3207   Dec 22 08:55   cat
-rw-rw-r-- 1   dick   18371  Jan 2 18:47    ch5
-rw-rw-r-- 1   dick   1009   Jan 5 15:31    ch7.accn
-rw-rw-r-- 1   dick   1466   Jan 5 15:34    ch7.bkup
-rw-rw-r-- 1   dick   23807  Jan 5 15:38    ch7.inf
-rw-rw-r-- 1   dick   627    Jan 3 18:30    ch7.stat
-rw-rw-r-- 1   dick   9756   Jan 5 14:05    chapt2
-rw-rw-r-- 1   dick   21570  Jan 5 14:41    chapt4
-rw-rw-r-- 1   dick   36678  Jan 6 20:39    chapt5
-rw-rw-r-- 1   dick   10362  Jan 5 15:28    chapt6
-rw-rw-r-- 1   dick   597    Dec 23 15:16   df
-rw-rw-r-- 1   dick   2915   Dec 21 21:14   diff

UNIX -> $
```

Единственное отличие этой команды от команды без флага —a в том, что в других случаях первые две строчки не были бы напечатаны. Заметим, что, если убрать флаг l, напечатаются только имена. Несколько флагов в одной команде ls следует использовать с осторожностью.

(2) Теперь взглянем на тот же список, что и в (1), но упорядоченный по времени последней модификации файлов. Файлы, модифицированные позже, появятся в списке раньше других.

```
польз. →   ls —tl <r>
UNIX →

total 829
-rw-rw-r--1    dick   36678   Jan 6 20:39    chapt5
-rw-rw-r--1    dick   23807   Jan 5 15:38    ch7.inf
-rw-rw-r--1    dick   1466    Jan 5 15:34    ch7.bkup
-rw-rw-r--1    dick   1009    Jan 5 15:31    ch7.accn
-rw-rw-r--1    dick   10362   Jan 5 15:28    chapt6
-rw-rw-r--1    dick   21570   Jan 5 14:41    chapt4
-rw-rw-r--1    dick   9756    Jan 5 14:05    chapt2
-rw-rw-r--1    dick   627     Jan 3 18:30    ch7.stat
-rw-rw-r--1    dick   18371   Jan 2 18:47    ch5
-rw-rw-r--1    dick   597     Dec 23 15:16   df
-rw-rw-r--1    dick   3207    Dec 22 08:55   cat
-rw-rw-r--1    dick   2915    Dec 21 21:14   diff
-rw-rw-r--1    dick   5705    Dec 21 15:35   ar
drwxrwxr-x 2   dick   704     Dec 20 13:48   book

UNIX -> $
```

Мы видим, что информация из примера (1) представлена в другом порядке. Команда «ls —tl» полезна, если мы пытаемся найти недавно измененные файлы. Можно одновременно задавать несколько флагов. Требуется, однако, следить, чтобы флаги не противоречили один другому. Так, мы не должны пытаться задавать одновременно флаги «t» и «u», поскольку они сортируют список по-разному.

Выводы.

Команда ls описывалась в гл. 4 только с флагом —l. В некоторых случаях она бывает полезна и для выдачи другой дополнительной информации.

7.4.2. ПЕЧАТЬ И УСТАНОВКА ВРЕМЕНИ

Команда: date
Синтаксис: date [ггммддччмм [.сс]]
Действие: большинство пользователей дают эту команду для получения времени*. Кроме того, администратор системы имеет возможность установить нужное время при запуске и перезапуске системы.
Аргументы: аргументы состоят из следующих полей:

гг год (если год не меняется, данное поле можно не задавать)
мм месяц (значение от 01 до 12)
дд день (значение от 01 до 31 в зависимости от месяца)
чч часы (24-часовое суточное время)
мм минуты (от 00 до 59)
сс секунды (от 00 до 59)

Примеры.

(1) Команда выполняется всякий раз, когда пользователь желает узнать время.

```
польз.  →  date <r>
UNIX    →  Sat Jan 10 14:51:02 EST 1981
        →  $
```

Эту команду может давать любой пользователь в любое время.

(2) Начальная установка и изменение времени производятся администратором системы. Некоторым пользователям также предоставляется такое право.

```
польз.  →  date 8101101816 <r>
UNIX    →  Sat Jan 10 18:16 EST 1981
        →  $
```

---

* В UNIX параметры даты (год, месяц, день) и времени дня (часы, минуты, секунды) всегда фигурируют вместе, поэтому при переводе используется термин «время» для обозначения полного астрономического времени. — *Примеч. ред.*

Выводы.

Комаида date несколько отличается от других комаид UNIX. Она, возможно, и не поиадобится слишком часто, но весьма полезиа. (Нередко мы желаем узиать время, не имея при себе часов.)

7.4.3. КТО РАБОТАЕТ В СИСТЕМЕ

Команда: who
Синтаксис: who [файл] [am I]
Действие: команда выдает список всех пользователей, работающих в настоящее время в системе, и имена связанных с ними терминалов.
Аргументы: комаида имеет два аргумента. Первый — имя файла с учетной информацией о текущих пользователях. По умолчанию это системный файл «/etc/utmp». Можно использовать и другой системиый файл «/usr/adm/wtmp», содержащий сведения о всех пользователях, входивших в систему с момента создания даииого файла. Второй аргумент, «amI», позволяет сообщить, кто же вы есть на самом деле, т. е. под каким именем вы вошли в систему.

Примеры.

(1) Дадим комаиду без аргументов.

```
польз.  →   who <r>
UNIX    →   darrin ttyb Jan 11 08:32
        →   dick ttya Jan 11:09:44
        →   pat ttyh Jan 11 14:12
        →   $
```

Выданиая иифорация включает имя пользователя, термииал и время входа в систему.
(2) Далее посмотрим, что сообщит нам команда с аргументом «am I».

```
польз.  →   who am i <r>
UNIX    →   dick tty1 Jan 17 11:22
        →   $
```

Выводы.

Комаида who служит для связи с некоторым пользователем. Предварительиио иадо проверить, работает ли пользователь в данный момент в системе. Кроме того, вы можете уточнить имя пользователя или узнать, за каким термииалом ои работает.

7.4.4. ПОЛУЧИТЬ ИМЯ ТЕРМИНАЛА

Команда: tty
Синтаксис: tty

Действие: эта команда печатает имя терминала, за которым вы работаете.
Флаги: нет.

Пример.

Поскольку команда используется в единственной форме, рассмотрим только один пример:

```
польз.  →   tty <r>
UNIX    →   ttyx
        →   $
```

Выводы.

Проще всего узнать имя терминала при помощи команды tty. Хотя вы можете получить ту же информацию и по команде «who am I».

7.4.5. ИМЯ ТЕКУЩЕГО КАТАЛОГА

Команда: pwd
Синтаксис: pwd
Действие: эта команда печатает полное имя вашего текущего (рабочего) каталога.
Флаги: нет.

Пример.

Имеется только одна форма команды:

```
польз.  →   pwd <r>
UNIX    →   /rlg/dick/book
        →   $
```

Выводы.

Эта команда очень удобна, когда вы хотите узнать, где находитесь. Подробно она рассматривалась в гл. 4.

7.4.6. СОСТОЯНИЕ ПРОЦЕССОВ

Команда: ps
Синтаксис: ps [флаг...] [файл]
Действие: эта команда предоставляет информацию о текущих активных процессах в системе. В зависимости от заданного флага выдаваемая информация может быть более или менее полной и относиться либо ко всем процессам, либо только к процессам, управляемым терминалами*.

---

* По определению, процессом управляет тот терминал, чей специальный файл открывает процесс первым среди других терминальных специальных файлов. Существуют процессы, не управляемые никаким терминалом. К ним относится процесс с идентификатором 1, выполняющий начальную программу раскрутки системы init, а также процессы, активизированные в стартовом командном файле системе /etc/rc. — *Примеч. ред.*

Флаги: команда имеет трн флага, которые могут задаваться вместе нли по отдельностн:

a выдается информацня о всех процессах, управляемых терминаламн;

x выдается ннформацня о всех процессах, не управляемых терминаламн. Обычно такие процессы вызываются системой;

l выдается полная информация, с указанием состояния каждого процесса. Подробно она опнсывается в прнмерах.

Примеры.

(1) Сначала попробуем дать команду без флагов.

```
польз.  →  ps ⟨r⟩
UNIX    →  PID   TTY     TIME        CMD
        →  203   co      0 : 03       —sh
        →  741   co      0 : 35    te status
        →  751   ?       ⟨defunct⟩
        →  752   co      0 : 00     ed
        →  753   co      0 : 03      ps
        →  $
```

Из прнмера видно, что нам предоставлена ннформацня о состояниях процессов:

PID — идентнфнкатор процесса;

TTY — номер термннала (co — операторская консоль);

TIME — суммарное время выполнения процесса;

CMD — команда, выполняемая процессом (т. е. что он делает).

Идентнфикатор процесса необходнм, если мы сделалн что-то не так и собираемся уннчтожнть процесс (см. команду kill). Имя команды сообщает вам, что делает данный процесс. Например, о первом процессе сообщается, что он вызван интерпретатором shell, так как прн входе в систему вы обычно попадаете под управленне программы shell. О следующем процессе сообщается, что выполняется команда «te status». Эта команда вызвана с одним аргументом status. Два другнх процесса определяются выполненнем самой команды ps, прнчем эта команда вызвана нз редактора ed.

(2) Теперь давайте выполним команду ps с флагом «a» для того, чтобы получить ннформацню о всех пользователях.

```
польз.  →  ps  a ⟨r⟩
UNIX    →  PID   TTY     TIME        CMD
        →  1494  co      0 : 01
        →  27    1       0 : 00      —x
        →  15    ?       1 : 07
        →  21    lp      0 : 00
        →  28    2       0 : 00      —0
        →  1553  5       0 : 00
        →  30    6       0 : 00      —x
        →  852   7       0 : 08
        →  1700  co      0 : 01
        →  1706  co      0 : 22
        →  1713  ?       ⟨defunct⟩
```

```
→   1714   co      0:00
→   1715   co      0:04      .sp
→   $
```

Флаг a в команде ps позволяет нам получить такую же информацию о всех активных пользователях, как и об индивидуальном пользователе.

(3) Теперь попробуем получить информацию о всех активных процессах (не только наших) и более подробной форме.

```
польз.  →   ps axl <r>
UNIX    →
```

F	S	UID	PID	PPID	CPU	PRI	NICE	ADDR	SZ	WCHAN	TTY	TIME	CMD
3	S	0	0	0	147	0	20	57	4	7670	?	1196:56	swapper
1	S	0	1	0	0	30	20	377	12	10574	?	0:02	
1	S	4	1494	1	0	30	20	323	16	10630	co	0:01	
1	S	0	27	1	0	28	20	365	12	107574	1	0:00	- x
1	S	0	13	1	0	40	20	351	12	164000	?	1:10	
1	S	1	15	1	0	40	20	332	28	164000	?	1:07	
1	S	0	21	1	0	29	20	341	12	110264	1p	0:00	
1	S	0	28	1	0	28	20	362	12	110104	2	0:00	- 0
1	S	0	1553	1	0	28	20	173	12	110022	5	0:00	
1	S	0	30	1	0	28	20	354	12	110166	6	0:00	- x
1	S	6	852	1	0	28	20	262	16	110332	7	0:08	
1	S	0	1700	1494	0	30	20	127	16	11224	co	0:01	
1	S	0	1706	1700	4	26	20	74	40	56356	co	0:32	
1	Z	0	1719	1706	5	50	20	0	0		?	<defunct>	
1	S	0	1720	1706	0	30	20	123	16	11350	co	0:00	
1	R	0	1721	1720	173	60	20	214	24		co	0:05	.sp

```
→ $
```

Мы видим, что три флага снабжают нас значительно большим количеством информации, чем в предыдущих примерах. Разъясним смысл всех колонок слева направо:

F внутренний флаг, связанный с процессом (01: процесс в памяти, 02: системный процесс, 04: процесс зафиксирован в памяти (физический ввод-вывод), 10: процесс выгружен, 20: процесс трассируется другим процессом));

S состояние процесса (0: не существует, S: задержан, W: ожидает, R: работает, 1: промежуточное состояние, Z: завершен, T: остановлен);

UID идентификатор пользователя — владельца процесса;

PID идентификатор процесса (используется в команде kill);

PPID идентификатор процесса, породившего данный процесс;

CPU системная составляющая приоритета процесса;

PRI приоритет процесса; большее число означает низший приоритет;

NICE пользовательская составляющая приоритета процесса, изменяется по команде nice;

ADDR для резидентного процесса — адрес в памяти, в противном случае — адрес на диске;

SZ размер образа процесса в блоках;

WCHAN событие, которого ожидает процесс с состоянием S или W; пустое поле означает, что процесс работает;

TTY терминал, управляющий процессом;
TIME суммарное время выполнения процесса;
CMD команда, выполняемая процессом.

Выводы.

Команда ps обычно нужна администратору системы при возникновении необычных ситуаций, таких, как переполнение таблицы процессов, потеря управления программой со стороны пользователя и т. п. Иногда она требуется и обычному пользователю, если он желает прекратить начатую работу. Это можно сделать, получив от команды ps идентификатор процесса (PID) и затем уничтожив его командой kill. Дополнительные сведения о команде ps будут изложены в гл. 9.

7.4.7. СВЕДЕНИЯ ОБ ИСПОЛЬЗОВАНИИ ДИСКА

Команда: du
Синтаксис: du[—s][—a][имя...]

```
пользз. ->  du /rlg/dick<r>
UNIX    ->  24      /rlg/dick/.calendars
        ->  479     /rlg/dick/bin
        ->  12      /rlg/dick/.directories
        ->  532     /rlg/dick/awssim/onyx/interp
        ->  16      /rlg/dick/awsslm/onyx/h
        ->  36      /rlg/dick/awssim/onyx/common
        ->  41      /rlg/dick/awssim/onyx/lib
        ->  643     /rlg/dick/awssim/onyx
        ->  1233    /rlg/dick/awssim
        ->  9       /rlg/dick/.ticklers
        ->  166     /rlg/dick/simdoc
        ->  43      /rlg/dick/working
        ->  392     /rlg/dick/onyx
        ->  274     /rlg/dick/reldata
        ->  30      /rlg/dick/source/new/yard/libr
        ->  89      /rlg/dick/source/new/yard/yard
        ->  123     /rlg/dick/source/new/yard
        ->  124     /rlg/dick/source/new
        ->  1       /rlg/dick/source/staging
        ->  90      /rlg/dick/source/stdio
        ->  224     /rlg/dick/source
        ->  25      /rlg/dick/library
        ->  333     /rlg/dick/book
        ->  136     /rlg/dick/atlas.tests
        ->  135     /rlg/dick/templet
        ->  2       /rlg/dick/play-work
        ->  89      /rlg/dick/newbook
        ->  302     /rlg/dick/emap.man
        ->  3932    /rlg/dick
        ->  $
```

Действие: эта команда сообщает о количестве блоков, занятых каждым файлом, и общее количество блоков для всех файлов. Информация выдается для всех файлов текущего каталога и рекурсивно для всех его подкаталогов.

Флаги: Команда имеет два флага. Флаг — s выводит только общее количество блоков для всех файлов. Флаг —a печатает информацию для каждого файла.

Примеры.

(1) Сначала дадим команду без флагов (см. программу на с. 115).

В данном примере мы получили количество блоков для каждого каталога и общее количество для всех каталогов.

(2) Посмотрим на действие флага —a.

```
пользз. ->   du -a /rlg/dick/book<r>
UNIX    ->   13    /rlg/dick/book/appdx.a
        ->   27    /rlg/dick/book/chapt.1
        ->   23    /rlg/dick/book/chapt.2
        ->   19    /rlg/dick/book/chapt.3
        ->   17    /rlg/dick/book/chapt.4
        ->   21    /rlg/dick/book/chapt.5
        ->   16    /rlg/dick/book/chapt.6
        ->   16    /rlg/dick/book/chapt.7
        ->   29    /rlg/dick/book/chapt.8
        ->   14    /rlg/dick/book/chapt.9
        ->   2     /rlg/dick/book/doit
        ->   38    /rlg/dick/book/steve.mac
        ->   7     /rlg/dick/book/table.n
        ->   1     /rlg/dick/book/title.n
        ->   32    /rlg/dick/book/comm
        ->   37    /rlg/dick/book/comm.fmt
        ->   16    /rlg/dick/book/status
        ->   329   /rlg/dick/book
        ->   $
```

Перечислены все файлы каталога book вместе с числом занятых блоков слева от имени. Мы видим, что в совокупности файлы каталога book занимают 329 блоков на диске.

(3) Теперь попробуем дать команду с флагом —s.

```
польз.  →   du —s /rlg/dick/book <r>
UNIX    →   329 /rlg/dick/book
        →   $
```

В данном случае получена только общая сумма.

Выводы.

Команда du полезна в тех случаях, когда на диске мало места и вы желаете знать, какие файлы и каталоги занимают больше всего блоков. Эта команда еще раз будет рассмотрена в гл. 9.

7.4.8. СВЕДЕНИЯ О СВОБОДНЫХ БЛОКАХ НА ДИСКЕ

Команда: df
Синтаксис: df [файловая система]
Действие: эта команда выводит количество блоков, доступных в заданной файловой системе. Обычно она используется администратором; в некоторых случаях, когда вы чувствуете, что в системе может не хватить памяти на диске, проверьте, сколько осталось свободных блоков (см. гл. 9 об администраторе системы UNIX).
Флаги: нет.

Пример.

В вашей конфигурации UNIX две файловые системы. Они имеют имена rp1 и rp3. Для определения числа свободных блоков в файловой системе rp3 вы просто даете команду

```
польз.  →   df /dev/rp3 ⟨r⟩
UNIX    →   /dev/rp3 6346
        →   $
```

В этом случае файловая система rp3 содержит 6346 свободных блоков, которые еще можно отвести под данные.

Выводы.

Как мы видим, нужно знать имя файловой системы до получения информации о свободной области. Кроме того, мы должны были указать, что это специальный файл (информация о нем располагается в каталоге /dev).

7.4.9. ОПРЕДЕЛЕНИЕ ТИПА ФАЙЛА

Команда: file
Синтаксис: file имя. . .
Действие: команда несколько раз проверяет каждый заданный файл и пытается определить его тип. Это может быть текст в коде ASCII, объектный файл, программа на языке Си и т. д.
Флаги: нет.

Пример.

Рассмотрим несколько случаев и проанализируем результаты.

```
польз.  →   file prog.c prog.o status doit ⟨r⟩
UNIX    →   prog.c: c program                  (программа на Си)
        →   prog.o: executable                 (выполняемый)
        →   status: roff, nroff, or eqn input  (входные данные для roff, nroff
                                                 или eqn)
        →   doit: commands                     (команды)
        →   $
```

У иас есть четыре файла. Первый — программа иа языке Си, второй — объектиый файл, третий — файл для nroff и четвертый — командиый файл UNIX. В этом примере мы получили правильную информацию. Однако в иекоторых случаях иифорнация может быть неточиой или даже иеправильиой.

Выводы.

Комаида file иужиа, когда вы не увереиы в содержимом файла и желаете избежать ошибки при его использовании.

7.4.10 ПЕЧАТЬ КАЛЕНДАРЯ

Команда: cal
Синтаксис: cal [месяц]год
Действие: эта команда печатает календарь для заданного месяца года или для всего года. Год обязательно должеи быть задаи, причем в пределах от 1 до 9999, т. е. если вы задали 81, то будет подразумеваться 81 год, а вовсе ие 1981 г.
Аргументы: имеется одии иеобязательный аргумеит, задающий месяц, со зиачением от 1 до 12.

Примеры.

(1) Выведем вначале калеидарь иа 1981 г. Даем команду

```
польз. →  cal 1981⟨r⟩
UNIX →
```

Далее см. с. 119.
(2) Теперь посмотрим, как узнать калеидарь некоторого месяца (см. с. 120).

Выводы.

Команда cal (как и комаида date) позволяет быстро получить информацию о месяце или годе.

7.4.11. ВОПРОСЫ

1. Как узнать, в каком месте файловой системы вы находитесь?
2. Как определить, сколько свободных блоков осталось в файловой системе usr?
3. Вы оставили свои часы дома и хотите узиать время. Как это сделать при помощи ЭВМ?
4. Как узнать имя, под которым вы вошли в систему?
5. Выведите идентификаторы активных процессов всех пользователей.
6. Напечатайте календарь иа 1981 г.

7.5. УПРАВЛЕНИЕ ТЕРМИНАЛОМ

Комаиды управления термииалом позволяют узнать состояиие управляющей иифорнации для даииого термииала и измеиить

Jan

S	M	Tu	W	Th	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Feb

S	M	Tu	W	Th	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

Mar

S	M	Tu	W	Th	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Apr

S	M	Tu	W	Th	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

May

S	M	Tu	W	Th	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Jun

S	M	Tu	W	Th	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Jul

S	M	Tu	W	Th	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Aug

S	M	Tu	W	Th	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Sep

S	M	Tu	W	Th	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

Oct

S	M	Tu	W	Th	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Nov

S	M	Tu	W	Th	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Dec

S	M	Tu	W	Th	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

ее, если этого требует ваш терминал. Существует много различных типов терминалов, и система должна уметь приспосабливаться к данному типу терминала.

7.5.1. УСТАНОВКА ФУНКЦИЙ ТЕРМИНАЛА

Команда: stty

Синтаксис: stty [аргументы . . .]

Действие: эта команда позволяет узнать состояние любого терминала в системе и настроить его на требуемый режим работы. Предоставляется возможность установить такие свойства, как скорость ввода-вывода, четность и др.

ПОЛЬЗ. -> cal 1 1981<r>
UNIX ->

Jan 1981

S	M	Tu	W	Th	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

UNIX -> $

Аргументы: для установки или отмены возможностей ввода-вывода в команде могут быть использованы следующие аргументы:

even включить контроль на четность;
—even выключить контроль на четность;
odd выключить контроль на нечетность;
—odd выключить контроль на нечетность;
raw включить прозрачный режим ввода;
—raw выключить прозрачный режим ввода;
nl концом строки считать только символ «перевода строки»;
—nl концом строки считать также символ «возврата каретки»;
echo отображать на экране каждый введенный символ;
—echo не отображать на экране вводимые символы;
lcase преобразовывать прописные буквы в строчные;
—lcase не преобразовывать прописные буквы в строчные;
tabs оставить символы табуляции;
—tabs заменить табуляции на пробелы при выводе;
ek установить принятые по умолчанию символы # для стирания и @ для отмены;
erase установить следующий за «erase» символ в качестве символа стирания;
kill установить следующий за «kill» символ в качестве символа отмены;
cr[0123] выбрать тип задержки после вывода символа возврата каретки;

nl[0123] выбрать тип задержки после вывода символа перевода строки;
tab[0123] выбрать тип задержки после вывода символа горизонтальной табуляции;
ff[01] выбрать тип задержки после вывода символа перевода формата;
bs[01] выбрать тип задержки после вывода символа возврата на шаг;
tty33 установить режим для терминала Teletype Corporation Model 33;
tty37 установить режим для терминала Teletype Corporation Model 37;
vt05 установить режим для терминала DEC T05;
tn300 установить режим для терминала GE TecmiNet 300;
ti700 установить режим для терминалов серии TI 700;
tek установить режим для терминала Tektronix 4014;
hup разорвать связь при последнем закрытии терминального файла;
— hup не разрывать эту связь;
0 разорвать связь немедленно;
50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb установить скорость передачи данных в бодах (если это возможно).

Примеры.

(1) Давайте сначала проверим состояние нашего терминала.

```
польз. →  stty ⟨r⟩
UNIX  →  speed 1200 baud
      →  erase = '#'; kill = '@'
      →  even odd —nl echo —tabs tab2 ff1
      →  $
```

Система сообщает о следующих параметрах режима:
1) скорость передачи данных 1200 бод;
2) символом стирания служит #;
3) символом отмены служит @ ;
4) установлен контроль четности;
5) установлен контроль нечетности;
6) возврат каретки означает перевод строки;
7) включено эхо-отображение символов;
8) символы табуляции заменяются пробелами;
9) включена задержка после вывода символа табуляции;
10) включена задержка после вывода символа перевода строки.

При необходимости мы можем изменить эти параметры режима терминала, указав альтернативные параметры режима в аргументах команды stty.

(2) Теперь посмотрим, как установить необходимые параметры режима терминала.

```
польз. →  stty 9600 >/dev/tty2 ⟨r⟩
UNIX  →  $
```

Если вы желаете убедиться в том, что скорость 9600 бод действительно установлена, дайте команду «stty>/dev/tty2», и она сообщит вам функции терминала. Здесь требуется знать номер терминала, который вы устанавливаете. Обычно этот формат команды используется при установке печатающего терминала.
(3) Другой необходимой функцией команды является установка символа стирания. При входе в систему обычно действует принятый по умолчанию символ стирания #. Однако для терминалов с экраном гораздо удобнее символ «возврат на шаг». Для его установки нужно дать команду

польз. → stty erase «возврат на шаг» ⟨r⟩
UNIX → $

На терминале это будет выглядеть так, как будто вы не ввели никакого символа стирания, поскольку символ «возврат на шаг» относится к управляющим (неграфическим) символам.

Выводы.

Команда stty необходима, если вы получили новый терминал и желаете установить для него требуемые функции. В некоторых случаях вам понадобится также изменить скорость передачи данных для удаленного ввода или для печати.

7.5.2. УСТАНОВКА ТАБУЛЯЦИИ

Команда: tabs
Синтаксис: tabs [аргументы]
Действие: эта команда позволяет установить функцию табуляции для различных терминалов, перечисленных при описании аргументов команды stty. По умолчанию установка соответствует большинству терминалов со скоростью 300 бод.
Аргументы: команда имеет два аргумента.

— n — используется, когда левый отступ не выравнивается должным образом;

терминал — этот тип терминала должен быть известен системе, которая производит необходимую установку функции табуляции.

Пример.

Для установки табуляции на терминале DIABLO 1620 дадим команду

польз. → tabs 1620 ⟨r⟩
UNIX → $

Система выполнит необходимую установку.

Выводы.

Если ваш терминал не попадает в спнсок терминалов, имеющих свои аргументы в команде stty, вы должны обратиться к администратору снстемы, н он сам подключнт терминал к системе

7.5.3. ВОПРОСЫ

1. У вас есть печатающее устройство, н его скорость установлена на 9600 бод. Как изменнть скорость на 1200 бод? Имя печатающего устройства tty3, имя вашего терминала tty1.

2. В чем преимущество аргумента tabs в команде stty?

3. У вас есть терминал DIABLO 1620. Как установнть для него функцию табуляции?

8. ИНТЕРПРЕТАТОР SHELL

До сих пор мы либо использовали команды UNIX по одной, либо соединяли их посредством программного канала или временного файла. Теперь обсудим командные файлы, содержащие несколько команд UNIX и служащие для выполнения команд точно так, как если бы мы вводили их последовательно одну за другой.

В некоторых случаях нам приходится периодически вводить одну и ту же последовательность команд для выполнения определенной задачи. Очевидно, что в этих случаях удобно ввести команды один раз в некоторый командный файл и затем вызывать его по мере необходимости. Командные файлы UNIX не только дают возможность автоматически запускать команды, но и позволяют управлять выполнением команд с помощью простого набора средств, типичных для языков программирования. Командные файлы называются иногда процедурами интерпретатора Shell, так как выполняются чаще всего под управлением этой программы UNIX. Во всем остальном они похожи на обычные файлы.

8.1. ПРОСТЫЕ КОМАНДНЫЕ ФАЙЛЫ

Если вы часто пользуетесь одной и той же простой командой для выполнения некоторой задачи, то у вас нет проблем. Однако, если команда усложнилась или требуется несколько команд, следует подумать о командном файле. Другая причина необходимости командных файлов — желание обеспечить пользователя простой методологией для выполнения сложной задачи без детализации ее частей.

Командный файл состоит из одной или нескольких команд UNIX. Он создается с помощью редактора путем обычного ввода команд с терминала. Разница между ним и обычным файлом лишь в том, что командный файл может быть выполнен произвольное число раз просто путем ввода его имени с терминала.

Создадим небольшой командный файл, который будет выводить список файлов некоторого каталога вместе со счетчиками числа строк, слов и символов для каждого файла. Для того чтобы выполнить такую команду непосредственно с терминала, нужно ввести «cd work; wc *». Для использования ее в виде командного файла мы должны сначала создать файл count, содержащий команды «cd work; wc*», для чего воспользуемся редактором ed.

После того как файл создан, мы можем выполнить его двумя различными способами. Первый способ состоит в выдаче команды sh, за которой следует имя комаидного файла. Это вызовет выполнение комаид в файле, как если бы они поступили прямо с терминала. Второй способ состоит в изменении режима доступа файла на «выполняемый» с помощью команды chmod.

Пример.

```
польз.  →  ed count ⟨r⟩
ED.     →  ? count
польз.  →  a ⟨r⟩
        →  cd work; wc* ⟨r⟩
        →  .⟨r⟩
        →  w ⟨r⟩
ED.     →  7
польз.  →  q ⟨r⟩
UNIX    →  $
```

Здесь мы создали командный файл count. Он напечатает количество строк, слов и символов для всех файлов каталога work.

Теперь давайте выполним count с помощью команды sh:

```
польз.  →  sh count ⟨r⟩
UNIX    →   7   21  83   file1
        →   7   21  83   file2
        →   7   21  83   file3
        →   1   5   11   table1
        →   1   5   11   table2
        →   6   6   34   x
        →   29  79  305  total
        →  $
```

Если мы с помощью команды «chmod 0777 count» изменим режим файла на выполняемый, то команду sh вызывать не иужно.

```
польз.  →  count ⟨r⟩
UNIX    →  7   21  83   file1
        →  7   21  83   file2
        →  7   21  83   file3
        →  1   5   11   table1
        →  1   5   11   table2
        →  6   6   34   x
        →  29  79  305  total
        →  $
```

Этот командиый файл удобен только если результат иеобходим для каталога work, поэтому не обладает достаточной гибкостью. В следующем разделе мы увидим, как с помощью аргументов сделать командные файлы более мощными.

8.1.1. КОМАНДНЫЕ ФАЙЛЫ И АРГУМЕНТЫ

Воспользуемся примером из предыдущего раздела для расширения возможностей командиого файла, причем имя обрабатываемого каталога будет задаваться при помощи аргумента. Это несколько напоминает использование аргументов в отдельных командах UNIX.

Аргумент в командном файле представляется двумя снмволами $n, где n — цифра от 1 до 9. Такнм образом, в одном командном файле может быть до девяти аргументов.

В предыдущем разделе мы подсчнтывалн статнстнку для файлов только в каталоге work. Теперь благодаря нашим знанням об аргументах мы можем предоставить пользователю возможность выбора необходнмого ему каталога.

Для этого нужно заменнть имя каталога work на аргумент $1, т. е. теперь мы должны задавать имя каталога во время вызова командного файла. Наш новый командный файл будет содержать «cd $; wc *». Для выполненння его даем следующую команду:

```
польз.  →   count work ⟨r⟩
UNIX    →    7  21  83  file1
        →    7  21  83  file2
        →    7  21  83  file3
        →    1  5   11  table1
        →    1  5   11  table2
        →    6  6   34  x
        →   29  79  305 total
        →   $
```

Результат тот же самый, но мы уже моглн выбирать имя каталога во время вызова командного файла. Теперь расшнрнм наш командный файл так, чтобы он позволял подсчнтывать чнсло строк, слов и снмволов только для файлов, заданных во время вызова командного файла. Наш новый файл будет содержать «cd $1; wc $2». Теперь мы должны задавать уже два аргумента. Первый всегда будет означать нмя каталога, второй — файл нлн шаблон для именн файла, о котором нужно получнть ннформацню. Мы можем выполннть его следующим образом:

```
польз.  →   count work file? ⟨r⟩
UNIX    →    7  21  83  file1
        →    7  21  83  file2
        →    7  21  83  file3
        →    21 63  249 total
        →    $
```

Мы запроснлн только файлы с нменамн, начинающнмнся с «file», за чем следует однн дополннтельный произвольный снмвол.

8.1.2. ВЛОЖЕННЫЕ КОМАНДНЫЕ ФАЙЛЫ

В UNIX можно вкладывать один командный файл в другой. Аргументы в этом случае обрабатываются так же, как в обычном командном файле.

Пример.

Создаднм главный командный файл. Он будет управлять дейстнем нескольknx локальных командных файлов; в данном случае у нас будет четыре таких файла. Каждый нз ннх управляет функ-

цнямн некоторого каталога. Главный командный файл используется для запуска всех остальных файлов. Конечно, можно вызвать любой локальный файл независимо от главного, но гораздо удобнее вызывать нх все одновременно.

В данном прнмере локальные командные файлы выводят спнсок файлов из своего каталога и число строк, слов н символов для каждого файла.

shell1 содержит:	shell2 содержит:	shell3 содержит:
wc file1	wc filea	wc filex
wc file2	wc fileb	wc filey
wc file3	wc filec	wc filez
в каталоге dick/A	в каталоге disk/B	в каталоге disk/C

Главный командный файл содержит:

```
master contains:
cd dick/A
echo shell1
shell1
cd../B
echo shell2
shell2
cd../C
echo shell3
shell3
echo done
```

Для выполнения главного командного файла мы должны установить режнм выполненння всех локальных командных файлов и убеднться в том, что нмеем право на нх выполненне (т. е. что они прннадлежат нам). Если файлы не выполняются, их можно вызывать только с помощью команды sh. Данный главный файл можно было бы сделать более гнбким путем введення аргументов для имен каталогов.

8.2. КОМАНДНЫЕ ПЕРЕМЕННЫЕ

Еще одно важное средство управления командныmн файламн — (командные) переменные. Переменным прнсваиваются значения в внде строк снмволов. Имена переменных должны состоять только из букв, цифр н символов подчеркнвання. Имя не может начинаться с цнфры. Строкн-значення не должны содержать пробелов.

Пример.

Напишем сначала простую процедуру, выводящую на термнал некоторые сообщения. Процедура (print) выглядит следующим образом:

```
A=first__time
B=second__time
C=third__time

echo $A
echo $B
echo $C
```

```
echo that's all folks
```

```
польз.  →  sh print (r)
UNIX    →  first__time
UNIX    →  second__time
UNIX    →  third__time
UNIX    →  that's all folks
```

Переменной можно присвоить значение строки с полным именем файла.

В следующем примере мы перейдем в другой каталог, выполним там процедуру print и вернемся назад в текущий каталог. Пусть файл print расположен в каталоге на один уровень выше текущего каталога current. Создадим процедуру с именем xx:

```
Path1=../
Path2=current

cd $Path1
sh print
cd $Path2
echo finished
```

Теперь выполним процедуру xx и посмотрим на результат.

```
польз.  →  sh xx (r)
UNIX    →  first__time
UNIX    →  second__time
UNIX    →  third__time
UNIX    →  that's all folks
UNIX    →  finished
```

Выполнение командных файлов возможно в других каталогах. Важно также отметить, что можно использовать переменные, значения которых равны заданным полным нменам файлов.

Следующие командные переменные имеют специальное значенне для интерпретатора Shell и не должны выступать в качестве обычных переменных.

$MAIL При интерактивной работе с интерпретатором Shell последний, перед тем как вывести приглашение, анализирует файл, заданный данной командой переменной. Если этот файл был изменен с момента предыдущего просмотра, Shell печатает сообщение «you have mail» («для вас есть почта») и только затем выдает приглашение для ввода следующей команды.

$HOME Это аргумент команды cd по умолчанию. Файлы, имена которых начинаются не с «/», отыскиваются в текущем каталоге. Каталог HOME определяется в файле /etc/passwd. Прн входе в систему пользователь попадает в этот каталог.

$PATH Список каталогов, содержащих команды (т. е. последовательность поиска команд). Предоставляет возможность задать список каталогов, где будет отыскиваться выполняемая команда (файл). По умолчанию команду ищут в текущем каталоге, затем в /bin и /usr/bin.

Обычно эти специальные переменные устанавливаются в файле .profile, расположенном в начальном каталоге пользователя. Он автоматически выполняется всякий раз, когда пользователь входнт в систему. Здесь вы можете задать выполнение любых начальных действнй. Автоматнческое выполнение этих действий позволяет избежать ошибок и упущений, которые возможны при явном выполнении действнй с терминала.

Пример.

Установим процедуру командного языка, автоматически выполняющуюся при каждом нашем входе в систему. Определим ее действия:

1) поиск команд сначала в каталоге /usr/dick/bin, затем /bin и /usr/bin;

2) замена функции «возврат на шаг» с символа '#' на символ возврата на шаг (backspace).

Командный файл должен иметь имя .profile, иначе снстема не будет вызывать его автоматически.

```
stty erase
echo backspace is backspace key
PATH=:/usr/dick/bin:/bin:/usr/bin
```

Переменная PATH задает не только список имен каталогов, но н последовательность поиска. В данном случае первым будет просматриваться /usr/dick/bin.

Команда stty не печатает символ возврата на шаг, поскольку это управляющий символ, но он устанавливается этой командой.

8.3. ВЫВОДЫ

Мы только в двух словах описали возможности интерпретатора Shell. Этого тем не менее вполне достаточно для решения большинства простых (и чаще всего встречающихся) задач. Как только вы освоите описанные средства, переходите к более сложным процедурам.

8.4. ВОПРОСЫ

1. Что такое командный файл и каково его значение?

2. Сколько аргументов может иметь командный файл? Как они обозначаются?

3. Приведите пример вложенных командных файлов.

4. Какие командные переменные имеют специальное значение?

9. АДМИНИСТРАТОР СИСТЕМЫ

9.1. ВВЕДЕНИЕ В СИСТЕМУ

Администратор системы — это человек, ответствеииый за обеспечеиие ежедневиого иормальиого функциоиироваиия машииы. Он обязаи запустить систему и проследить за завершеиием ее работы, регистрировать иовых пользователей и удалять выбывших, обеспечить сохраниость файловой системы иа магнитиой леите и ее восстановление в случае повреждеиия, запуск учетных программ, сообщающих, кто (и в каком количестве) использует время процессора и память иа диске. Одним словом, в его обязанности входит поддержаиие иормальной работы системы.

Администратор, кроме того, ведет системный журнал, в котором отражается все, что связаио с запуском и остановкой системы, измеиеииями коифигурации аппаратуры, обиаружеииыми ошибками устройств и файловой системы.

9.1.1. ПРИВИЛЕГИРОВАННЫЙ ПОЛЬЗОВАТЕЛЬ

В системе есть два особых пользователя с правами и возможиостями, ие доступными обычиым пользователям. Одии из них иазывается «привилегированным пользователем» и имеет имя root. Ои задается при входе в систему. Одиако термии «привилегироваииый пользователь» (superuser) в UNIX имеет более общий смысл. Привилегироваииый пользователь может делать в файловой системе и в UNIX вообще практически все, что ему угодио. Другой пользователь, с именем bin, не обладает всеми возможностями привилегироваииого пользователя, но является владельцем миогих важиых системных файлов и может поступить с ними так же, как и привилегироваииый пользователь.

Привилегированный пользователь действительио обладает особыми полиомочиями. Его ие касаются правила защиты файлов (за исключеиием иескольких тривиальиых и иесущественных случаев).

Привилегированиый пользователь имеет право читать и писать файлы и каталоги, принадлежащие любому пользователю. Однако, как сказал мудрец: «Чем выше власть, тем выше ответственность». Привилегироваиный пользователь может в результате простой ошибки разрушить всю файловую систему, так что опе-

рационная система UNIX не будет работать вообще. Например, привилегированный пользователь мог бы легко уничтожить систему следующим образом:

```
польз.  →  chdir /bin ⟨r⟩
        →  rm * ⟨r⟩
UNIX    →  #
```

Если это случится, ни одна системная программа не сможет работать, в том числе и программы восстановления системы. Но, положим, вы все же сделали такую ошибку. Тогда следует остановить систему, восстановить главную файловую систему с ленты защиты и доложить своему руководителю о случившемся. Даже самые высококвалифицированные и опытные привилегированные пользователи допускают огрехи, так что не стесняйтесь признавать свои ошибки. Если вы будете следовать процедурам, излагаемым в данной главе (особенно касающимся защиты файловой системы в случае ее разрушения), то сможете без особых потерь восстановить нормальную работу системы даже в случае самых непрнятных ошибок.

Заметим, между прочим, что вместо обычного прнглашения UNIX — символа $ — здесь появился символ #. Этот символ — приглашение для привилегированного пользователя. Он напоминает вам, что вы действительно являетесь таковым и обладаете неограниченными правами. Ввиду большой ответственности, накладываемой на привилегированного пользователя, число людей, знающих его пароль, рекомендуется уменьшить до двух — это администратор системы, и еще один человек, которому этот пароль нужен только в исключительных случаях (например, в отсутствие администратора).

Есть две возможности стать привилегированным пользователем. Первая — войти в систему под именем root и правильно указать пароль. Вторая — запустить программу su под управлением интерпретатора Shell. В этом случае вы также должны дать пароль. Для выхода из привилегированного режима наберите на клавиатуре ctrl/d и вы либо выйдете нз системы (если вошли в нее под именем root), либо вернетесь к обычному режиму (если вы давали команду su). При загрузке системы и работе в однопользовательском режиме вы всегда будете обладать правами привилегированного пользователя (этот случай рассматривается далее в разделе «Загрузка системы»).

Пример.

Вход в режим привилегированного пользователя:

```
UNIX    →  login:
польз.  →  root ⟨r⟩
UNIX    →  password:
польз.  →  ⟨r⟩        ввод пароля, эхо-отображение отсутствует
UNIX    →  #           приглашение для привилегированного пользователя
```

Для использования команды su вы уже должны находиться в системе.

UNIX	→	$	обычное приглашение
польз.	→	su ⟨r⟩	
UNIX	→	password:	
польз.	→	⟨r⟩	ввод пароля, эхо-отображение отсутствует
UNIX	→	#	приглашение для привилегированного пользователя

Другой, особый пользователь имеет имя bin. У него нет тех привилегий, что у пользователя root, однако он владелец всех системных программ в каталогах /bin и /usr/bin, а также всех драйверов устройств (специальных файлов), таких, как терминалы, печатающие устройства и т. д. Вообще говоря, все системные файлы принадлежат этим двум пользователям.

Администратор системы должен уметь войти в систему с именами пользователей bin и root. Если вы в состоянии выполнить необходимую работу под именем bin (не прибегая к привилегированному режиму root) — делайте это. Чем меньше времени вы работаете в привилегированном режиме, тем меньше вы допустите случайных ошибок, ведущих к разрушению системы. Чтобы войти в систему под именем bin, следует выполнить те же действия, что и для пользователя root.

При чтении данного раздела вам может показаться, что мы с недоверием относимся к вашей способности успешно работать в системе, поскольку постоянно подразумеваем совершение ошибок в тех или иных случаях. Хорошо известен «неприятный закон Мэрфи»: «если может произойти ошибка, она происходит». Прекрасные программисты в прошлом совершали глупейшие ошибки, часто из-за переоценки своих способностей. По-видимому, неприятности неизбежны, несмотря на опыт людей, работающих с системой, а зачастую и вследствие этого опыта. Именно поэтому мы настоятельно советуем постоянно быть готовыми к тому, что система может разрушиться.

9.1.2. РЕГИСТРАЦИЯ НОВЫХ ПОЛЬЗОВАТЕЛЕЙ

Обратим внимание на несколько простых задач, для выполнения которых не требуется специальных знаний. Сначала рассмотрим задачу включения нового пользователя в систему.

Полный перечень пользователей системы хранится в файле /etc/passwd, называемом «учетным файлом пользователей». Каждая строка этого файла содержит запись об одном пользователе. Например, первые несколько строк типичного файла /etc/passwd могут содержать следующее (см. с. 133).

Каждая строка состоит из полей, разделенных двоеточием и обозначающих: имя пользователя; зашифрованный пароль пользователя (не сам пароль); идентификатор пользователя; идентифи-

```
root:ty2IuAdu:0:1:Super-User:/:
su::0:1:Super-User:/:
daemon::1:1:System:/usr/1115:
bin::3:1:System:/bin:
opr::3:1:Operator:/usr/adm:
dump::3:1:System:/usr/adm:/usr/adm/dumper
games::26:1:Games:/usr/games:
dick:nnAnXu4n:92:2:Dick Gauthier:/rlg/dick:
roy:jjtx62bk:68:2:Roy Oishi:/rlg/roy:
lea:NYr2Njzk:11:5:Lea Gallardo:rlg/lea:
```

катор группы; имя и фамилию пользователя (или любой другой комментарий); начальный каталог пользователя — каталог, в который он попадает при входе в систему; имя интерпретатора команд пользователя. Если последнее поле отсутствует, то нужен обычный интерпретатор команд Shell.

В приведенном примере dump будет псевдопользователем. При входе в систему он запускает программу /usr/adm/dumper; как только программа завершается, dump выходит из системы.

Идентификатор пользователя и группы — это числа от 0 до 255. Каждый пользователь должен иметь отдельный идентификатор; связанные между собой пользователи (например, работающие над одним проектом) должны иметь один и тот же идентификатор группы. Учетный файл групп, аналогичный /etc/passwd, имеет имя /etc/group.

Для включения нового пользователя нужно просто добавить строку в файл /etc/passwd. Новая строка будет похожа на остальные, поле пароля должно быть пустым. Когда пользователь в первый раз войдет в систему, он вызовет программу passwd для установки своего пароля (см. гл. 2). Эта программа изменит соответствующим образом файл /etc/passwd путем занесения туда зашифрованного пароля пользователя.

```
польз.  →  ed /etc/passwd <r>          вы должны иметь право записи
ED.     →  1234
польз.  →  $ <r>                       установка на конец файла
польз.  →  a <r>                       вход в режим добавления
польз.  →  sam::12:1:Sam Jones:/usr/sam: <r>
польз.  →  <r>                         выход из режима добавления
польз.  →  w <r>                       запись измененного файла
ED.     →  1264
польз.  →  q <r>
UNIX    →  #
```

Если вы в качестве начального задали еще не существующий каталог (так обычно и делается), его следует создать прежде, чем новый пользователь в первый раз войдет в систему. (Для этого существует команда mkdir, описанная в гл. 7.) Измените имя владельца каталога с root на имя нового пользователя, как это описано в следующем разделе.

```
польз.  →   cd /usr ⟨r⟩          установка корневого каталога пользователей
UNIX    →   #
польз.  →   mkdir sam ⟨r⟩        имя должно совпадать с именем в файле
                                 паролей
UNIX    →   #                    вы должны иметь разрешение на запись
```

9.1.3. СМЕНА ВЛАДЕЛЬЦА И КОДА ЗАЩИТЫ ФАЙЛА

Еще одна простая функция включает изменение владельца и кода защиты файла. Менять владельца файла может только привилегированный пользователь. Команда смены владельца довольно проста:

chown имя-пользователя имя-файла

где имя-пользователя — имя, которое задается при входе в систему; имя-файла — имя файла, меняющего владельца.

Аналогично можно изменить принадлежность файла к некоторой группе.

chgrp имя-группы имя-файла

где имя-группы — одно из имен, перечисленных в файле /etc/group.

Пример.

```
польз.  →   chgp 1 sam⟨r⟩
UNIX    →   #
```

Код защиты файла представляет собой девять битов информации, соответствующих разрешению читать, писать и выполнять файл. Такое разрешение отдельно предоставлено для владельца файла, пользователей, входящих в группу, к которой принадлежит файл, и прочих пользователей. Таким образом, всего имеется девять разрешений. Если мы дадим команду «ls —1», то увидим эти биты разрешений слева на экране терминала.

```
польз.  →   ls  —1 ⟨r⟩
UNIX    →
```

```
-rwxr-xr-x   1 dick   25102   Apr 6 15:11    3dcomp
-rw-rw-rw-   1 dick    7548   Sep 16 15:37   iii
-rwsr-sr-x   1 root    3322   Apr 8 1976     passwd
drwxr-xr-x   2 dick     256   Sep 21 15:55   sacourse
drwxrwxr-x   2 root     512   Dec 18 09:24   book
-rw-r--r--   1 dick      83   Oct 15 17:03   file1
-rw-r--r--   1 dick      83   Oct 19 13:31   file2
-rw-r--r--   1 dick      83   Oct 19 13:44   file3
-rw-r--r--   1 dick      83   Oct 22 18:43   filen
-rw-r--r--   1 dick      82   Oct 19 12:18   sfile1
-rw-r--r--   1 dick      83   Oct 15 17:05   sfile2
-rw-r--r--   1 dick      82   Oct 19 13:47   sfile3
-rw-r--r--   1 dick      11   Oct 15 15:45   table1
-rw-r--r--   1 dick      11   Oct 15 15:45   table2
-rw-rw-r--   1 dick      11   Oct 15 15:45   table3
-rw-rw-r--   1 dick      11   Oct 15 15:45   table4
```

Для большинства представленных здесь файлов их владелец (dick) может чнтать н писать файлы, а члены группы и прочие пользовател н — только читать данные файлы. Файл iii могут чнтать н пнсать: владелец, член группы н прочие пользовател. Файл 3dcomp — это программа, которую может выполиять любой пользователь. Файл sacourse — каталог, что отмечено буквой d слева от битов защиты. Не следует путать букву d с битами защнты. Она не принадлежит к таковым, н файл нельзя сделать каталогом путем нзменення кода защиты. Комаида ls просто указывает с помощью буквы d, что файл является каталогом.

Существует еще одна возможность доступа к файлу. Программы со включенным битом «смены ндентнфикатора пользователя» будут выполияться с эффектнвным ндентификатором пользователя — такнм же, как и у владельца файла. Значит, если у привилегированного пользователя есть программа, требующая при выполнении особых условий, и если он желает, чтобы все прочие пользовател моглн выполнять данную программу, то должен установить для нее бнт «смеиы ндентифнкатора пользователя». Тогда прн любом запуске программа будет обладать правами привилегированиого пользователя. Для прнмера рассмотрим прнведенную выше программу passwd, служащую для измеиеиия пароля пользователя. Она требует установкн бита «смены ндентнфнкатора», чтобы нметь возможность изменять файл /etc/passwd, прннадлежащий пользователю root. Если обычиый пользователь выполняет программу passwd, то времеино получает все права пользователя root. После завершеиия программы он вступает в свои обычиые права.

Заметим, кстати, что в выводиой ннформацнн комаиды «ls —l» бит смены идеитификатора пользователя обозиачается буквой s в позицин бита x. Это может ввестн в заблужденне — на самом деле бит смены ндентифнкатора отделеи от бнта выполиения. Программа ls просто считает, что s удобно поместнть именио в это место (автор программы ls, по-виднмому, счел, что налнчне s имеет смысл только прн включенном бите x).

Значеиие бнтов защнты в каталогах отличается от значений в файлах. Бит x для каталога означает «разрешение понска»; если пользователю не разрешен понск в каталоге, то он не может перейти в иего по комаиде cd н не имеет доступа ни к одиому файлу этого каталога. Разрешение записи для каталога означает возможность добавлять, удалять н переимеиовывать файлы в этом каталоге. Таким образом, разрешенне удалять файл не завнсит от разрешення пнсать в иего или делать его пустым. Программам пользователя (в том числе прнвилегированиого) не разрешается пнсать в каталог, как в обычный файл. Разрешеиие чтения для каталога действует, как н для обычного файла, — позволяет читать каталог. Так, программа ls чнтает каталогн с

целью получення нмен файлов. Заметнм, что разрешенне читать файл не завнсит от разрешення чнтать его нмя*.

Для нзменення кода защнты файла нспользуется команда chmod:

chmod восьмеричное-чнсло имя-файла

Восьмернчное-число формнруется следующнм образом: цнфры слева направо относятся соответственно к владельцу, группе, прочнм пользователям. Каждая цнфра — сумма цифр 1 (для разрешения выполнення нли понска), 2 (для разрешення запнсн) и 4 (для разрешення чтення). Для установкн разрешення читать файл FILENAME всем, а запнсывать только вам как владельцу, дайте команду:

польз. → chmod 644 FILENAME ⟨r⟩
UNIX → $

В результате код защнты будет установлен в «-rw-r-r-».

Для установкн бнта смены ндентифнкатора пользователя необходнмо подставнть цнфру 4 слева к коду защнты. Цнфра 2 устанавлнвает бнт смены идентнфнкатора группы, аналогнчный биту смены ндентнфнкатора пользователя; 6 устанавливает оба указанных выше бита.

Устанавливать код защнты файла может только владелец файла илн прнвилегнрованный пользователь.

9.2. СОСТАВ СИСТЕМЫ

В этом разделе дается краткая характеристнка основных компонентов вашей вычнслнтельной снстемы: аппаратуры, разлнчных категорнй программного обеспечення, файловой снстемы.

9.2.1. АППАРАТНОЕ ОБЕСПЕЧЕНИЕ

Аппаратное обеспеченне снстемы состонт нз вычнслнтельной машины («компьютера»), одного нли нескольких дисковых устройств, на которых хранятся все данные снстемы, нескольких терминалов того нлн нного тнпа (телетайпы нли вндеодисплеи), печатающего устройства, на которое выводятся результаты, устройства для чтения н записи магнитных лент.

9.2.1.1. Вычислительная машина

Вычнслнтельная машина — главная часть системы. Она содержнт центральный процессор н основную память. Центральный процессор — активный компонент снстемы, выполняющнй про-

---

* Инымн словамн, чтобы получнть любой доступ к файлу, нужно нметь право чтення всех каталогов, указанных в нмени файла. Лишь после того, как прочитаны все такие каталоги, проверяются права доступа к самому файлу. — *Примеч. ред.*

граммы путем копирования их в память и с помощью машинных инструкций, составляющих программу. Кроме того, центральный процессор имеет консоль оператора — переднюю панель процессора с переключателями — посредством чего машина запускается и останавливается. Консоль оператора у каждой машины своя, поэтому для работы с ней нужно ознакомиться с соответствующей инструкцией.

9.2.1.2. Дисковые устройства

Каждая система имеет по крайней мере одно дисковое устройство, состоящее из одного или более магнитных дисков. Различные типы дисковых устройств содержат диски стационарные несъемные, вертикально расположенные съемные, горизонтально расположенные съемные. Один диск может содержать от 2,5 до 100 мегабайтов в зависимости от типа устройства (мегабайт — миллион байтов или символов). Скорость передачи информации (чтения или записи данных) также меняется от модели к модели. Каждая модель дискового устройства имеет свой собственный набор лампочек и кнопок, однако большинство из них представляет собой комбинации следующих индикаторов:

READY (готово) эта лампочка включается при работе с системой;

FAULT (ошибка) включается в случае серьезной аппаратной ошибки — может потребоваться ремонт устройства;

LOAD (загрузка) включение этой лампочки означает, что можно снять диск;

PROT (защита) означает, что на диск нельзя записывать информацию; для отмены защиты следует нажать соответствующую клавишу (возможно, содержащую лампочку).

Диски используются для хранения всех данных системы, в настоящий момент не обрабатываемых процессором. Это относится не только к данным в программах, но и к самим программам. Когда вы вызываете программу для выполнения, UNIX отыскивает файл с заданным именем, читает его с диска в память и выполняет.

Следует сохранять в чистоте диски и устройства, так как они очень чувствительны: одна пылинка, попавшая в устройство, может вывести диск из строя. Поэтому не рекомендуется курить, а также есть в машинном зале.

Дисковые устройства обычно нумеруются 0,1,... с тем, чтобы процессор мог их различать. Когда у вас более одного устройства, удобно (если можно) по мере надобности менять их номера (см. раздел об использовании защитной копии UNIX). На одних типах устройств это можно сделать просто путем переключения кнопки с номером устройства. На других типах требуется переключение кабеля.

9.2.1.3. Терминалы

Терминал — основное средство контакта между пользователями и вычислительной машиной. В вашем распоряжении находится несколько типов терминалов. Терминалы бывают: с лучевой трубкой (телевизионным экраном), с «твердой копией», т. е. непосредственно печатающий данные на бумагу; может быть терминал типа Diablo с высококачественной печатью, которая «не похожа на выдачу машины» и пригодна к непосредственному использованию в качестве деловых документов. Ваш терминал, возможно, будет иметь переключатель «online/offline» или «online/local». Он должен быть установлен в позицию «online» для связи с машиной. Кроме того, различные терминалы работают с различной скоростью, регулируемой переключателем на терминале. Для определения типа и скорости каждого присоединенного к системе терминала существует файл с соответствующей информацией. Ведение этого файла входит в обязанности администратора системы и будет рассмотрено в последующих разделах.

9.2.1.4. Печатающее устройство

Печатающее устройство — это обычно главное устройство, на которое выводятся окончательные результаты работы программы. Подключено устройство к машине или нет показывает кнопка «online/offline» и соответствующий индикатор. Если установлен режим «offline» («автоном»), устройство не напечатает ничего, но машина знает, что устройство отключено и подождет, пока не будет установлен режим «online» («работа»). Вы можете нажать кнопку «top-of-form» («перевод формата») в режиме «offline» и устройство пропустит одну страницу бумаги.

Существует несколько индикаторов ошибок, таких, как «paper» («бумага»), «gate» («замок»), «ribbon» («лента»), загорающихся в случае неготовности устройства по физическим причинам (для их выяснения, возможно, придется поднять крышку устройства). В таких случаях обычно на передней панели устройства также загорается индикатор «alarm» («не готово»). Установите причины ошибки и затем нажмите кнопку «не готово» — это вызовет установку устройства в нормальное состояние. Затем нажмите кнопку «работа» для связи устройства с машиной — устройство должно начать (либо продолжить) печать с минимальными потерями выходных данных.

Может быть и так — устройство на некоторое время прекращает печатать без видимых причин. Ситуация «зависания» печатающего устройства возникает по вине программного обеспечения, а не аппаратуры. Вы можете за полминуты вывести устройство из этого состояния, воспользовавшись любым терминалом UNIX (см. раздел 9.5.4).

9.2.1.5. Магнитная лента

На магнитной ленте удобнее всего хранить данные большого объема, если они в настоящее время не требуются. Например, на лентах можно держать копию дисковой файловой системы для предотвращения ее потери в случае разрушения диска. Кроме того, ленты — это наилучшее средство для переноса данных или программ с одной машины на другую.

Ленточные устройства могут работать с плотностью записи 800 байтов/дюйм (т. е. символов на дюйм) или 1600 байтов/дюйм. Некоторые устройства могут переключаться с одной плотности на другую, но всякая лента должна читаться с той плотностью, с которой она была записана. Источником несовместимости является также число дорожек. Различают 7-дорожечные и 9-дорожечные ленточные устройства, причем ни одно из этих устройств не может читать ленту, записанную на устройстве другого типа. Большинство установок UNIX могут читать и писать только 9-дорожечные ленты.

Устройства также различаются по скорости чтения и записи данных, но это не оказывает принципиального влияния на правильность работы с лентой. Скорости записи и чтения одной и той же ленты могут различаться.

Все ленточные устройства работают приблизительно по одному принципу. Когда вы хотите что-либо записать на ленту, следует установить на обратной стороне катушки пластиковое кольцо (не без основания называемое «кольцом записи»). Если же, напротив, нужно избежать записи на нее, кольцо следует снять. Ленточное устройство не в состоянии записывать на ленту, не имеющую кольца записи.

После того как вы установили или сняли кольцо записи, поставьте ленту на свободное устройство. Отмотайте часть ленты и заправьте ее в принимающую катушку так, как это показано на диаграмме, расположенной на передней панели устройства. Проверьте точность намотки ленты и нажмите кнопку «load» («загрузка»). Лента начнет двигаться вперед в поиске «точки загрузки», представляющей собой кусочек фольги, наклеенной в начале ленты. Рабочая часть ленты всегда начинается в этой точке. В зависимости от конкретной модели устройства теперь, возможно, придется нажать кнопку «on-line» или что-то в этом роде для того, чтобы загорелся индикатор «on-line». После этого лента будет готова к работе с программой.

После завершения работы с лентой следует нажать кнопку «rewind» («перемотка»). В зависимости от предыдущих операций над лентой, она либо начнет перематываться назад к точке загрузки, либо, если она уже была в этой точке, окончательно смотается с принимающей катушки. Следовательно, для полного освобождения ленты, возможно, придется нажать кнопку «перемотка» два раза.

9.2.2. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Программное обеспечение вычислнтельной системы — это набор программ, выполняющнх различные функцнн. Операцнонная система UNIX состонт нз двух частей. Первой н нанболее существенной частью является так называемое «ядро». Ядро выполняет только самые основные функцни операцнонной системы: оно управляет всемн операциямн ввода-вывода н распределяет между пользователямн время центрального процессора (по траднцни это называется «планнрованнем»).

Вторая часть содержит ннтерпретатор командного языка Shell и основной набор команд пользователя. Прн загрузке системы ядро отыскивается на днске, где оно постоянно хранится и читается в оперативную память машнны. После этого система может начинать работать. Ядро знает, где расположены остальные части программного обеспечения, и по требованию пользователя загружает и запускает их. Эти программы, однако, не входят в ядро.

Команды илн программы, доступные пользователю, представляют собой обычные файлы, расположенные в файловой системе, чаще всего в каталогах /bin нли /usr/bin. Позже мы обсудим более подробно состав н значение этих каталогов. Адмнннстратор системы может добавлять программы в эти каталогн, расширяя таким образом возможности системы. Ядро, в случае крайней необходимости (напрнмер, при установке новой аппаратуры или прн измененнн характера работы системы), может нзменять только специалнст, хорошо разбнрающийся в системе UNIX.

Наиболее важные параметры ядра UNIX:

• какой тнп днска (или его часть) используется для размещения главной файловой системы;

• какая область н на каком диске должна служнть в качестве области свопинга.

9.2.3. КРАТКИЙ ОБЗОР ФАЙЛОВОЙ СИСТЕМЫ

Файловая система — это концептуальное понятне UNIX, с которым адмнннстратор системы должен быть хорошо знаком. Она представляет собой организованную группу файлов, заннмающих весь диск или его часть (если это большой днск, превышающий максимальный размер файловой системы UNIX). Файловая система не может начинаться на одном диске, а заканчиваться на другом.

Существует одна, так называемая «главная файловая система», содержащая минимум средств, необходимых для работы UNIX. Другне файловые системы делаются доступными путем «монтирования» нх к главной файловой системе таким образом, что для пользователя они становятся обычными (хотя н довольно большими) каталогамн. Несмотря на то что фактически такие каталоги располагаются в отдельных файловых системах, непрнвнлегированный пользователь этого не замечает. Напрнмер, на вашей

установке каталог /usr и все его подкаталоги могут быть монтированной файловой системой, что позволяет достичь определенной гибкости при работе с ней. Вы можете пожелать, например, в разное время монтнровать разные файловые системы. Многие процедуры администратора выполняются в терминах всей файловой системы; сохранение файловой системы на ленте, проверка целостности информации на диске — наиболее важные из них.

9.3. ЗАПУСК И ОСТАНОВКА СИСТЕМЫ

Здесь поясняется, как запускать и останавливать UNIX, при этом всякий раз вы обязаны заносить необходимые записи в системный журнал.

9.3.1. ЗАВЕРШЕНИЕ РАБОТЫ С СИСТЕМОЙ

Ежедневно в конце рабочего дня вы будете обязаны останавливать систему и в начале следующего рабочего дня запускать снова. Иногда вам понадобнтся остановить систему по просьбе обслуживающего персонала машины. Следуя предлагаемой нами процедуре, гарантнрующей нормальное состояние файловой системы при остановке машнны, вы не доставите никаких неприятностей своим пользователям.

Прежде всего вы должны войти в режим привилегированного пользователя (см. раздел 9.1). После этого следует сообщить всем пользователям о готовящейся остановке системы:

```
польз. →  /etc/wall ⟨r⟩
       →                  введите любое сообщение иа одиой или
       →                  иескольких строках
польз. →  ctrl/d
UNIX   →  #
```

Это сообщение будет послано всем работающим в данный момент пользователям. Еслн вы хорошо знакомы с процедурой остановки системы, то отметьте этот факт в файле /etc/motd, где собраны все сообщения текущего дня и который появляется на экране термннала всякий раз, когда пользователь входит в систему.

После уведомления пользователей об остановке системы следует с помощью команды who (печатающей список работающих пользователей) провернть, все ли пользователи кроме вас вышли из системы. Затем вы даете команду:

```
польз. →  ps alx ⟨r⟩
UNIX   →
```

```
F S UID   PID  PPID  CPU  PRI  NICE  ADDR  SZ   WCHAN  TTY  TIME  CMD
1 S   0     1     0    0   30    20    74  12   11556  ?    0:22  /etc/init
1 S  13  6908     1    0   28    20   135  16  107466  co   0:03  -sh
0 S   0    21     1    0   28    20   147  12  107632  1    0:00  - x
1 S   0     8     1    0   40    20    65  12  164000  ?    7:05  /etc/update
1 S   1    10     1    0   40    20   155  28  164000  ?    5:49  /etc/cron
1 S   0    19     1    0   40    20    71  12  164000  1p   0:01  /etc/openup
1 S  13  6675     1    0   28    20    61  16  107714  4    0:10  -sh
1 S   4  9214     1    0   30    20   141  16   12062  2    0:02  -sh
0 S   0    24     1    0   28    20   253  12  110060  5    0:00  -x
0 S   0    25     1    0   28    20   267  12  110224  6    0:00  -x
1 S   0    26     1    0   28    20   117  12  110370  co   0:00  -ps axl
```

→ #

Здесь не должно быть никакнх пользовательскнх процессов, т. е. никакнх работающнх команд, кроме /etc/update н другнх «следящих» процессов, нницинрованных файлом /etc/rc (он будет рассмотрен в следующих разделах). Если некоторые процессы пользователей еще не завершнлись, нужно либо убедиться в их скором завершеннн и подождать немного, либо_уннчтожить их с помощью команды kill.

Теперь следует выполннть три действия:

польз. → kill —1 1 ⟨r⟩ (вы должиы уиичтожить все процессы кроме init (процесс 1) и вериуть систему в однопользовательский режим)

польз. → sync ⟨r⟩

и иажать киопку «остаиов» на передней паиели машииы.

В тот момент, когда вы остановили центральный процессор, система окончательно завершнт работу. Теперь, если вы собираетесь передать снстему обслуживающему персоналу для ремонта нлн профнлактнки, следует позаботиться о защите содержнмого днсков от ошибочной запнсн на них другой информации. Еслн вашн дискн съемные — снимнте нх. В противном случае сохраните необходнмые данные на ленте. Обслуживающнй персонал обладает «замечательным талантом» затирать наиболее нужную информацию на дисках.

9.3.2. ЗАГРУЗКА СИСТЕМЫ

Загрузка системы предшествует началу работы машнны. Загрузка необходнма поскольку, когда машина начинает работать, еще нет ннкакой работающей программы, нет ее н в памяти машины. Поэтому первое, что нужно сделать — прочитать программу в память. Необходнмая программа (ядро операционной системы) находится где-то на днске и процессор, к сожаленню, не знает, где именно. Нужно передать процессору программу, которая сообщнт ему, как прочнтать в память ядро операцнонной системы. Эту программу нельзя прочитать с днска, поскольку процессор не знает, ни где она находится, нн как ее прочитать, из-за того, что в памятн пока ннчего нет, получается порочный круг.

Как же все-таки заставить систему работать? Загрузить* совсем маленькую программу, знающую, как прочитать собственно операционную систему. Эта процедура называется раскруткой** («поднять себя за собственные шнурки»).

Если вы остановили машину, как это было описано в предыдущем разделе, то можно начинать загрузку. Если произошел «крах системы», означающий, что из-за ошибки она была прервана в момент выполнения полезных действий, следует также остановить машину перед повторной загрузкой.

9.3.3. ПРОЦЕДУРА ЗАГРУЗКИ

После выполнения шагов, описанных в разделе «Загрузка системы», введите «boot» (а также либо перевод строки, либо возврат каретки — по соглашению). Вы увидите приглашение «:». Такая начальная процедура загрузки зависит от типа машины. Это может быть описанная выше процедура — простое нажатие кнопки.

Затем наберите команду в форме

```
xx(0,0)unix
```

где xx — двухбуквенный код UNIX, обозначающий тип диска, с которого загружается система (он отличается от двухбуквенного кода, принятого для обеспечения устройств в операционных системах фирмы DEC). Некоторые из кодов выглядят следующим образом:

```
rp . . ... . . . RP03
hp . . . . . RP04/5/6
rk . . . . . RK05
```

Когда вы получите приглашение «#», UNIX будет готова работать в однопользовательском режиме. Приступайте теперь к проверке целостности файловой системы, как это описывается ниже.

9.3.4. ПРОВЕРКА ФАЙЛОВОЙ СИСТЕМЫ. ПРОСТОЙ СЛУЧАЙ

При однопользовательском режиме необходимо выполнить процедуру проверки целостности файловой системы (обычно называемую chk). Эта процедура запускает программы проверки для каждой файловой системы. Программы проверки обнаруживают несоответствия в файловой системе, и сообщения о них появляются на экране терминала. Вот типичная процедура chk:

```
dcheck /dev/rrp1
icheck /dev/rrp1
dcheck /dev/rrk3
icheck /dev/rrk3
```

---

* — аппаратным способом. — *Примеч. пер.*

** В оригинале «bootstrapping» — шнуровка, затягивание шнурков. — *Примеч. пер.*

Такую процедуру следует выполнять ежедиевно, даже если система находится в нормальном состоянии, с тем, чтобы застраховаться от проблем, могущих возникиуть в будущем. Исправление файловой системы в ряде случаев будет трудиым делом, но если аппаратура в хорошем состоянии, такие случаи редкость. Сложные случаи мы рассмотрим позже, сейчас ваша задача — научиться определять исправна файловая система или нет. При этом желательно следовать рекомеидованиой выше процедуре остаиовки системы.

При условии, что все в порядке, выходные данные процедуры chk могли бы выглядеть следующим образом:

```
/dev/rrp1:
/dev/rrp1:

files 542 (r=448,d=47,b=13,c=34)
used 8779 (i=212,ii=7,iii=0,d=8553)
free 6586
missing 0
 /dev/rrp1:
 /dev/rrp1:

entries    link     cnt
976         0        0
981         0        0
984         0        0
999         0        0
1000        0        0
1001        0        0
1002        0        0
1003        0        0
1004        0        0
1005        0        0

/dev/rrp3:
/dev/rrp3:
files 1003 (r=934,d=69,b=0,c=0)
used 7564 (i=206,ii=0,iii=0,d=7358)
free 6834
missing 0
/dev/rrp3:
/dev/rrp3:
```

Общий формат вывода процедуры chk для каждой файловой системы: имя устройства этой системы (дважды); далее следуют данные об общем количестве файлов, о количестве специальиых больших, очень больших (это может отсутствовать) файлов, каталогов, косвеиных блоков, использованных и свободиых. Между двумя строчками с именем устройства /dev/ может располагаться

информация о неисправности файловой системы: количестве пропущенных блоков, дублированных блоков, дублированных в списке свободных блоков и т. п. (Подробнее см. раздел 9.8).

В некоторых вариантах системы UNIX имеется команда fcheck. Если она у вас есть, ее можно использовать вместо команд icheck и dcheck, вызываемых из вышеописанной процедуры chk. Если fcheck не задает вам вопросов — значит, все в порядке. Типичный вывод процедуры fcheck для нормальной файловой системы выглядит следующим образом:

Фаза 1 — проверка блоков.
Фаза 3 — проверка имен файлов.
Фаза 4 — проверка счетчиков числа связей.
Фаза 5 — проверка списка свободных блоков.
235 файлов 1713 блоков 2200 свободных блоков.

После того как вы устраните все неисправности в файловой системе, можно переводить UNIX в многопользовательский режим.

9.3.5. МНОГОПОЛЬЗОВАТЕЛЬСКИЙ РЕЖИМ

Для перехода в многопользовательский режим необходимо сначала установить в системе время. Это делается с помощью команды:

```
date yyMMddhhmm.ss
```

где устанавливаются следующие двухцифровые поля: yy — год, MM — месяц, dd — день, hh — час, mm — минута, ss — секунда. Поле секунд может быть опущено, тогда подразумевается 00. Год, месяц и день могут быть также опущены, если они не изменились с момента последней установки времени. Следовательно, обычно требуется ввести только часы и минуты, например:

```
польз. →  date 1321
UNIX  →  Mon Feb 16 13:21:58 PST 1981
```

Система печатает введенное время в форме удобной для того, чтобы его можно было проверить и при необходимости исправить.

Затем вы, возможно, захотите отметить в файле /etc/motd некоторую необходимую для пользователей информацию, например, об утерянных файлах. Для этого нужно просто записать сообщение в файл motd, как это делается для обычных файлов. Файл motd всегда находится в каталоге /etc.

Наконец, следует нажать клавишу ctrl/d. В течение нескольких секунд будет выполняться командный файл /etc/rc. Он монтирует требуемые файловые системы; все терминалы будут активизированы, а система готова к многопользовательской работе. Не следует забывать делать необходимые отметки в системном журнале.

9.4. РАСПРЕДЕЛЕНИЕ РЕСУРСОВ СИСТЕМЫ

Администратор системы должен следить за тем, чтобы все ресурсы системы были справедливо распределены между пользователями. Особо нужно следить за дисковой памятью и процессами.

9.4.1. ПАМЯТЬ НА ДИСКЕ

Память на диске — это такой ресурс, о котором пользователи не должны заботиться до тех пор, пока не произойдет переполнение диска.

Прежде всего вы должны знать, сколько необходимо оставить свободного места в файловой системе. Когда система устанавливается, следует приблизительно определить количество свободной памяти для каждой файловой системы и отмечать все случаи ее быстрого использования. Желательно около 15% каждой файловой системы оставлять свободными (если файловая система не сильно изменяется, то меньше, если изменяется, то больше). Так, каталог /tmp файловой системы, будучи пустым в начале работы, значительно заполняется и освобождается в течение дня. Поэтому в каталоге /tmp рекомендуется резервировать по крайней мере 100 блоков свободной памяти на каждого работающего пользователя.

Необходимо знать, с помощью каких программ можно обнаружить нехватку памяти. При запуске системы и выполнении процедуры chk (или fcheck, если она имеется), становится известно число свободных блоков для каждой файловой системы. Переполнение памяти на диске не позволит создавать новые файлы и увеличивать уже имеющиеся. С помощью команды df вы всегда можете определить, сколько осталось свободных блоков в любой файловой системе. Кроме того, есть еще один способ определить переполнение памяти. Если консоль начинает печатать строчки No space on dev M/m (нет памяти на устройстве M/m), где M и m — тип и номер устройства, значит, файловая система переполнена. Лучше этого не допускать ввиду неприятных последствий, таких, как нарушение целостности файловой системы и невозможность сохранения результатов редактируемых файлов пользователей.

Предположим теперь, что переполнение файловой системы все-таки произошло. Что нужно предпринять? Прежде всего, если нехватка памяти вызвана переполнением каталога /tmp, следует просто немного подождать — некоторые программы, требующие много памяти на диске, могут завершиться через небольшой промежуток времени. Среди программ, активно использующих каталог /tmp, можно выделить сортировку больших файлов, компиляцию, редактирование, обработку текстов. Если ситуация не меняется, следует попытаться удалить некоторые файлы из каталога /tmp. Для этой цели нужно дать команду «ls —lut/tmp», которая выведет список всех файлов /tmp, отсортированных по времени последнего

доступа к файлу. Вы увидите, какие файлы не потребовались в течение продолжительного времени и, следовательно, могут быть удалены. Если и это не поможет, придется заново запустить систему с полной очисткой каталога /tmp. Этот радикальный способ следует использовать лишь в крайнем случае, когда не дают результата никакие другие.

При переполнении памяти в обычной пользовательской файловой системе необходимо сначала выяснить, не создал ли кто-либо из пользователей (возможно, непреднамеренно) очень большой файл. Для этого подойдет команда /etc/wall. С ее помощью можно предупредить пользователей о недостатке памяти и попросить их удалить ненужные файлы. Если просьба останется без внимания, то можно применить административные меры. Для этого существуют команды du и find.

Команда du сообщает, сколько блоков на диске занимает данный каталог и все содержащиеся в нем файлы и подкаталоги. Эту команду можно использовать тогда, когда вы подозреваете какого-либо конкретного пользователя в слишком большом потреблении памяти на диске. Если же вы не знаете, откуда начать поиск нарушителя, то воспользуйтесь командой find. Она подробно описана в гл. 5. Здесь же мы отметим, что способность отыскивать файлы по заданным размерам, имени владельца, времени доступа делает ее мощным средством для обнаружения «пожирателей» дисков. Например, команда

```
польз. →  find usr —size +1000 —a —mtime —2 —a
       →  —exec ls —l '{}' ';' (r)
```

будет запускать «ls —l» для всех файлов в /usr бо́льших 1000 блоков, измененных за последние два дня. Это помогает точно определить, из-за чего произошло переполнение файловой системы.

Команда find также весьма полезна для поиска файлов, к которым не было доступа в течение продолжительного времени (например, двух месяцев). Часто, если вы обратите внимание владельцев этих файлов на такой факт, вам удастся убедить их либо удалить эти файлы, либо записать на ленту, и место на диске освободится. Запись файлов в архив на ленту производится программой tar (см. раздел 9.10).

9.4.2. ПРОЦЕССЫ

Интенсификация эксплуатации системы может привести к достижению предела памяти или числа процессов. Симптомы этих двух случаев схожи: сначала значительно увеличивается время реакции системы, а затем, в случае серьезных нарушений, может произойти остановка системы.

О достижении предела памяти мы узнаем при входе в так называемый режим «трешинга», когда операционная система начинает сама потреблять так много времени процессора, что до пользователей просто не доходит очередь. Фактически, планирую-

щие программы в этом случае требуют гораздо больше времени, чем программы, которым они должны это время предоставлять. Если такие ситуации возникают достаточно часто, следует заказать больше памяти для системы. С другой стороны, если вы вдруг обнаружили сильное замедление работы и предполагаете, что кто-то производит слишком много вычислений, можно использовать команду ps. Вы должны быть хорошо знакомы с форматом вывода команды ps, даже если все идет нормально. Но, положим, кто-то выполняет одновременно 10 программ nroff, тогда следует напомнить ему, что это сильно замедляет работу остальных пользователей.

Наконец, можно дойти до предела числа процессов. UNIX позволяет одновременно выполнять только ограниченное число процессов. Это число фиксируется при генерации системы. Если вы попытаетесь выполнить одновременно большое число процессов, система просто откажется создавать новые процессы, а интерпретатор Shell напечатает на терминале сообщение «try again» («повторите команду»). В этих случаях вам может понадобиться команда ps, чтобы посмотреть, не создал ли кто-нибудь случайно слишком много процессов. Если попытка дать команду ps также завершается сообщением «try again», то все, что вы способны сделать в этой ситуации, — предложить пользователям сохранить результаты редактирования и выйти из системы, причем число выполняющихся процессов может уменьшиться так, что появится возможность выполнить команду ps. Если и это не поможет, вы должны остановить систему путем остановки процессора, запустить ее заново и устранить неисправности файловых систем, вызванные остановкой системы. К счастью, редко кто-либо таким образом ошибочно переполняет таблицу процессов системы.

При необходимости вы можете перестроить ядро так, чтобы увеличить предел числа одновременно выполняемых процессов.

9.4.3. УЧЕТ

Имеются две стандартные программы UNIX, ведущие учет работы пользователей: ac, выполняющая учет входов в систему и количества использованного времени для каждого пользователя, и sa, анализирующая записи в журнале выполнения команд.

Любая программа генерирует в удобочитаемой форме статистическую информацию о своем выполнении. Собственно учет этой информации выполняется путем запуска учетных программ при входе пользователя в систему и выходе из нее. Учет ведется с помощью файла /usr/adm/wtmp. Если его нет, информация не учитывается.

Команда «/etc/accton /usr/adm/acct» в вашем файле /etc/rc будет обеспечивать такой учет.

Помните, что оба файла с учетной информацией имеют тенденцию к росту и периодически их следует чистить (например, раз в месяц).

9.5 ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ

В этом разделе рассматриваются некоторые дополнительные системные процедуры: файл /ets/rs, который запускается всякий раз, когда UNIX входит в многопользовательский режим; программа /ets/cron, выполняющая запуск необходимых программ в заданное время дня, недели или месяца; файл /etc/ttys, сообщающий UNIX типы подключенных терминалов; процедура «оживления» печатающего устройства; некоторые полезные командные файлы.

9.5.1. ФАЙЛ /etc/rc

Файл/ets/rc — это командный файл, автоматически запускающийся при входе UNIX в многопользовательский режим. Следует знать, что он делает, чтобы при необходимости можно было его модифицировать.

Файл /ets/rc обычно содержит три типа команд: команды монтирования, подсоединяющие пользовательские файловые системы к главной файловой системе; некоторые обслуживающие программы, такие, как команда «rm /tmp/*» для очистки каталога /tmp или учетные процедуры и, наконец, строки, запускающие так называемые «следящие программы»*. Следящая программа не принадлежит реально какому-либо пользователю, но периодически запускается во время работы UNIX с целью выполнения определенной функции. Примером такой программы может быть /etc/cron или /etc/update. Программа /etc/cron рассматривается ниже; программа /etc/update вызывает модификацию диска каждые тридцать секунд и должна работать на каждой системе.

9.5.2. ПРОГРАММА /etc/cron

Программа /etc/cron используется для запуска других программ в заданное время. Она читает данные из файла /usr/lib/crontab в заданном формате. Предположим, что у нас есть процедура /usr /adm/dumper, выполняющая ежедневный сброс системы на ленту, и мы хотим запускать ее в 8 часов вечера каждого рабочего дня. Соответствующая строка в /usr/lib/crontab выглядела бы следующим образом:

```
0 20 * * 1—5 /usr/adm/dumper
```

Эта строка говорит о том, что в 00 мин, 20 ч, каждый день каждого месяца, с понедельника по пятницу включительно, необходимо выполнять программу /usr/adm/dumper.

Но для того чтобы программа cron могла работать, она должна быть где-то запущена. Наилучшим местом вызова /etc/cron является файл /ets/rc.

---

* В оригинале «daemons» — дьяволы; этот же термин используется и в документации по UNIX. — *Примеч. пер.*

9.5.3. ФАЙЛ /etc/ttys

Этот файл содержит информацию о каждом терминале.

Каждая строка ttys описывает один порт.

Первый символ каждой строки может быть 0 или 1. Строки с нулем просто игнорируются. Следующий символ описывает тип терминала. Он зависит от реализации, но символ 0 всегда используется для коммутируемого терминала и для консоли. Оставшаяся часть строки представляет собой имя специального файла в каталоге /dev, относящееся к данному порту. Например, строка 10console указывает на специальный файл /dev/console.

Действительные физические порты располагаются на заднем щите вычислительной машины. К каждому выходу можно приклеить метку с буквой, соответствующей данному порту, с тем, чтобы можно было переключать терминалы с одного порта на другой.

9.5.4. МЕХАНИЗМ СИСТЕМНОЙ ПЕЧАТИ*

Время от времени в работе механизма системной печати могут возникать трудности (при наличии печатающего устройства). Мы рассмотрим, как этот механизм функционирует, и предложим процедуры устранения связанных с ним проблем. Системная печать — средство, позволяющее пользователям по очереди печатать файлы, не заботясь о том, в каком состоянии находится в данный момент печатающее устройство.

Пользователи для выполнения системной печати должны давать команду lpr, которая помещает в каталог /usr/lpd копию печатаемого файла и небольшой командный файл, указывающий необходимые при печати действия. После этого lpr запускает следящий процесс системной печати /etc/lpd.

Lpd прежде всего проверяет наличие файла lock в каталоге /usr/lpd. Если он там имеется, lpd завершается, причем предполагается, что в данный момент уже работает другая копия этой программы. В противном случае создается файл lock; таким образом, всегда работает не более одной копии lpd. После того как создан файл lock, осуществляется поиск соответствующего командного файла; затем он выполняется и удаляется. Если командных файлов больше нет, файл lock уничтожается и программа lpd завершается.

На некоторых установках печатающие устройства обладают способностью «зависать». Симптомы этого следующие: 1) отсутствует печать по команде lpr, 2) в каталоге /usr/lpd имеются файлы для печати и 3) печатающее устройство находится в состоянии готовности к работе с машиной. Для разрешения про-

---

* Иногда вместо термина «системная печать» употребляют слово спулинг — от английского SPOOLing (Simultaneous Peripherial Operation On-Line) — параллельное выполнение операций с периферийным устройством — *Примеч. пер.*

блемы сначала дайте команду «ps a». Найдите процесс, связанный с командой lpd или /etc/lpd, и запомните номер этого процесса. Выполните команду «kill — 3 n», где n — номер процесса печати. Печатающее устройство должно начать печатать. Если процессов с lpd нет, удалите файл lock и запустите системную печать снова:

```
польз. →   rm —f /usr/lpd/lock ⟨r⟩
польз. →   /etc/lpd ⟨r⟩
```

Если устройство не заработает, проверьте его простой командой:

```
польз. →   echo stuff > /dev/lp ⟨r⟩
```

Должно напечататься слово stuff и выполниться перевод страницы. Если и это не получится, скорее всего, неисправно само устройство.

Бывают ситуации, когда кто-то с помощью lpr направляет файл в системную печать, а затем решает, что файл все-таки не следует печатать. Чтобы избавиться от ненужной печати, прежде всего следует остановить печатающее устройство (перевести его в автономный режим). Затем нужно перейти в каталог /usr/lpd и дать команду «ls —l». Напомним, что lpd создает там два файла при каждом своем запуске: небольшой файл с командами для системной печати, именуемой буквами dfa и номером, и копию данных, которые нужно печатать, — этот файл именуется некоторыми буквами и тем же номером на конце. Попытайтесь определить, какой файл следует удалить по его размеру и имени владельца. Если необходимо, убедитесь в этом при помощи команды cat, прерывая вывод после нескольких первых строк, после чего удалите файл с данными и соответствующий ему файл dfa.

Если нежелательный файл уже начал печататься, можно также применять вышеприведенную процедуру. В конце концов, вы можете уничтожить программу системной печати, удалить файлы вместе с файлом lock и снова запустить lpd, переведя затем устройство в состояние готовности. Поясним эту процедуру на полном примере. Предположим, что пользователь Joe случайно дал команду:

```
польз. →   lpr ⟨bigmodule.o ⟨r⟩
```

вместо требуемой команды:

```
польз. →   lpr ⟨bigmodule.c ⟨r⟩
```

Объектный код — не самый лучший файл для печати, тем более что он обычно содержит символы перевода формата, вызывающие выбросы большого количества бумаги. При виде этого вы останавливаете устройство нажатием кнопки «offline» («автоном»). Затем идете к терминалу, где уже ранее вошли в систему и выполняете следующую процедуру:

```
польз.  →   chdir /usr/lpd ⟨r⟩
польз.  →   ls —1 ⟨r⟩
UNIX    →
total 6
-rw-rw-rw-    1 mary      226   Mar 17   03:28      cfa02324
-rw-rw-rw-    1 joe       843   Mar 17   03:25      cfa02326
-rw-rw-rw-    1 mary      63    Mar 17   03:28      dfa02324
-rw-rw-rw-    1 joe       63    Mar 17   03:25      dfa02326
-rw-rw-rw-    1 sue       41    Mar 17   12:29      dfa02329
— 1 daemon                0     Mar 17   12:14      lock
        →   $

польз.  →   rm [dc]fa02326 ⟨r⟩
польз.  →   ps a ⟨r⟩
UNIX    →   8:      1356 —0

        →           2284   —
        →   h:      393    — w
        →   i:      2314   ed chapt.2
        →   i:      12     —
        →   8:      2320   lpd
        →           2337   ps a
        →           2315   bin/sh recompile
        →   $

польз.  →   kill 2320 ⟨r⟩
UNIX    →   2320: not found
польз.  →   su ⟨r⟩
UNIX    →   Password:

польз.  →           ввод пароля, эхо-отображение отсутствует
UNIX    →   #       приглашение для привилегированного пользо-
                    вателя
польз.  →   kill 2320
UNIX    →   #
польз.  →   ctrl/d
UNIX    →   $       выход из привилегированного режима
```

Между символами # и $ вы вводите управляющий символ ctrl/d.

Теперь вы снова можете вернуть печатающее устройство в режим «online» («готово»). Оно напечатает несколько строк (освобождая буфер) и пропустит последнюю страницу. Вернитесь к терминалу и введите:

```
польз.  →   rm —f lock ⟨r⟩
польз.  →   /etc/lpd ⟨r⟩
UNIX    →   $
```

и печатающее устройство начнет вывод данных для пользователя Mary.

9.5.5. НЕКОТОРЫЕ ПОЛЕЗНЫЕ КОМАНДНЫЕ ФАЙЛЫ

Одна из привлекательных черт UNIX — возможность легко строить состоящие из выполняемых команд процедуры на языке Shell. Искусство программирования на командном языке позволяет создавать процедуры высокой сложности, сравнимые с про-

граммами на обычных языках программирования, но в то же время достаточно простые для непрограммистов, знакомых с командным языком. Составлять их поэтому можно столь же быстро, как вводить с терминала. Командные файлы удобны для ежедневных и еженедельных процедур защиты дисков на лентах, обслуживающих и учетных процедур, а также если нужно вводить стандартные последовательности команд. Наиболее важные и длинные процедуры рекомендуется оформлять с помощью командных файлов, что уменьшает вероятность ошибки.

Для создания командного файла нужно с помощью редактора просто поместить в файл команды, которые вы хотите выполнять. Сюда же можно включать комментарии, поясняющие действие. Комментарий вводится строкой с первым символом «:», вслед за чем идет пробел и собственно текст комментария. Этот же формат используется и для метки командного языка. После завершения ввода текста нужно изменить код защиты файла, установив биты выполнения. Рекомендуется устанавливать код 775, т. е. «rwxrwxr-x», позволяющий всем читать и выполнять файл, в то время как изменять файл сможете только вы либо члены вашей группы.

Вы можете задавать аргументы процедуре. Для этого существуют командные переменные $1, $2, ..., $9. Переменная $1 обозначает первый аргумент команды, $2 — второй и т. д. В качестве примера можно привести простую процедуру печати и удаление содержимого каталога:

```
польз. →  ls —l $1 > rmlist ⟨r⟩
польз. →  rm $1/* ⟨r⟩
```

Если этот файл назвать cleanout, то его можно вызвать для очистки каталога /tmp следующей командой:

```
польз. →  cleanout /tmp ⟨r⟩
```

В командном языке имеются также средства управления. Команда shift перенумеровывает аргументы, т. е. $2 становится $1 и т. д. Кроме того, можно организовывать циклы, обрабатывающие произвольное количество аргументов. Более сложная версия cleanout может быть такой*:

```
echo>rmlist
for d do
is —l $d>>rmlist
rm $d/*
```

Теперь вы можете одной командой очистить несколько каталогов.

```
польз. →  cleanout direct1 direct2 direct3
```

---

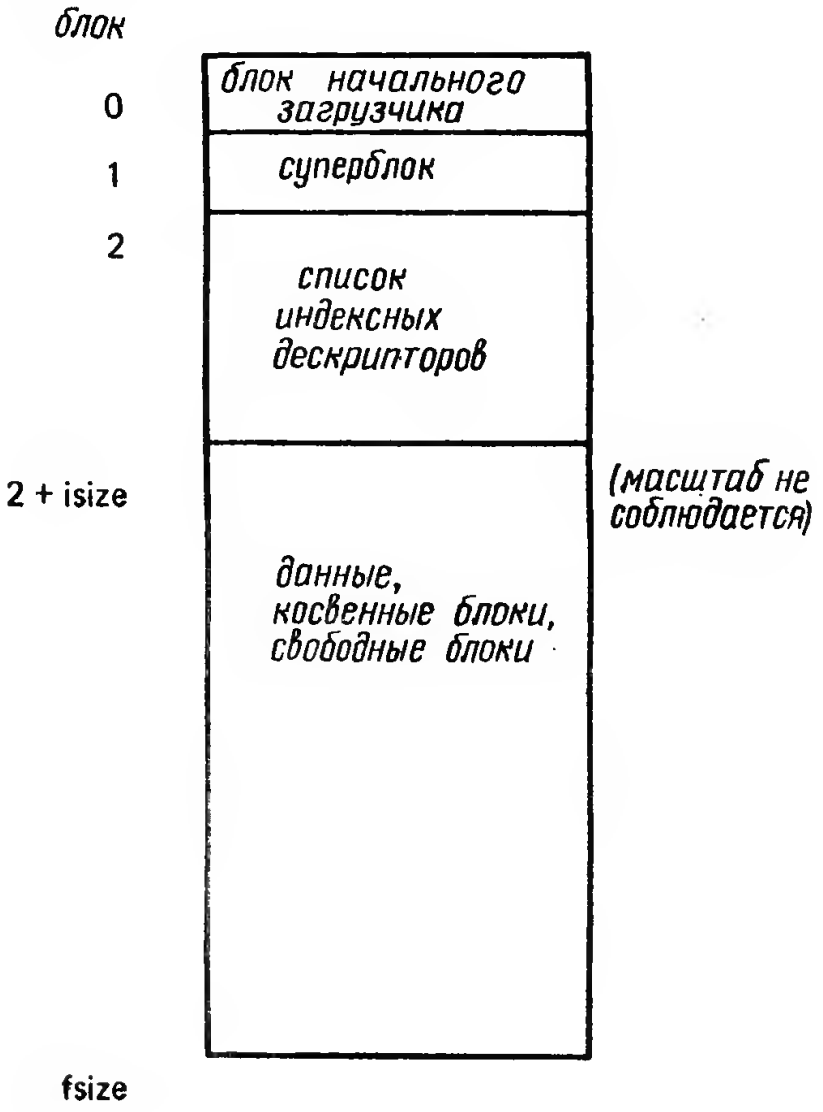
* Автор употребляет в этом примере средство UNIX, нигде в данной книге не описанное. Команды, расположенные между строками «for d do» и «done», будут выполнены в цикле столько раз, сколько аргументов задано в команде cleanout, причем переменная $d каждый раз будет обозначать очередной аргумент команды. — *Примеч. пер.*

9.6. БОЛЕЕ ПОДРОБНЫЕ СВЕДЕНИЯ О ФАЙЛОВОЙ СИСТЕМЕ

В предыдущих разделах мы часто упоминали файловую систему, так что у вас уже есть некоторое представление о том, что это такое и для чего она предназначена. Вы знаете, что файловая система — это набор файлов, что она помещается на одном диске (или несколько файловых систем на одном большом диске). Вам известно, что файловые системы присоединяются каким-то образом к главной системе, что их нужно беречь от разрушения, следить за тем, чтобы небольшие повреждения не привели к более серьезным разрушениям. В данном разделе мы более глубоко изучим вопросы, связанные с файловой системой.

9.6.1. СТРУКТУРА ФАЙЛОВОЙ СИСТЕМЫ

Приведем сначала общую структуру файловой системы. Отдельные ее части поясняются ниже.



UNIX рассматривает любой диск (или часть большого диска), содержащий файловую систему, как последовательность блоков по 512 символов (или байтов) каждый. Блоки обычно нумеруются 0, 1, 2, ... на каждом диске (или части диска). Каждая файловая система содержит следующие блоки:

1. Блок начального загрузчика (блок 0) нужен при загрузке системы UNIX, но не используется файловой системой.

2. Суперблок (блок 1) — это «заголовок» файловой системы. Он включает размер файловой системы (fsize), число блоков, отведенных под индексные дескрипторы (isize), начало списка свободных блоков (здесь дается упрощенная картина; полный формат суперблока можно найти в разделе FILSYS (5) руководства пользователя UNIX (Unix Programmer's Manual)).

3. Блоки с индексными дескрипторами. Для каждого файла в файловой системе имеется точно один индексный дескриптор (в блоке — 8 индексных дескрипторов). Индексный дескриптор содержит информацию о типе файла (обычный, специальный, каталог), владельце файла, коде защиты, количестве связей (напомним, что UNIX позволяет именовать файл в нескольких каталогах), размере файла, времени последнего доступа к файлу (чтения) и его модификации (записи), указателях местоположения файла на диске. Индексный дескриптор корневого каталога файловой системы имеет номер 2.

4. Блоки с данными. Здесь содержатся фактические данные, входящие в файлы. Неиспользованные блоки этой области связываются в список свободных блоков.

Помимо блоков с данными в этой области могут находиться так называемые косвенные блоки, содержащие номера других блоков. Если файл превышает определенный размер, индексный дескриптор указывает на один или несколько косвенных блоков, которые в свою очередь указывают на блоки с данными. Индексный дескриптор очень большого файла может иметь 10 указателей на блоки с данными, указатель на косвенный блок с указателями на блоки с данными, указатель на косвенный блок с указателями на косвенные блоки с указателями на блоки с данными и, наконец, указатель на тройной косвенный блок.

Индекс — номер индексного дескриптора, причем первый индексный дескриптор нумеруется 1, второй — 2 и т. д. Индекс файла можно рассматривать как внутреннее системное имя или идентификатор файла.

Индексные дескрипторы распределяются для файлов в порядке их доступности на диске. Поскольку в типичной системе UNIX файлы многократно создаются и удаляются, индексы достаточно хорошо разбросаны и их значения непредсказуемы. Для номеров блоков ситуация та же.

К вопросам, касающимся структуры файловой системы, близко примыкает и вопрос о формате каталога. В отличие от других систем UNIX хранит в каталоге только два элемента информации для каждого файла: его индекс и имя, под которым он из-

вестен в данном каталоге. Отсюда ясно, каким образом файл может иметь более одной связи — два каталога содержат записи с одинаковым индексом файла. Из этого также следует, что файлы UNIX не имеют внутренней связи со своими именами: имена содержатся в каталогах, а не в индексных дескрипторах. Вся же остальная информация о файле содержится в индексном дескрипторе. Таким образом, когда UNIX открывает файл с заданным именем в некотором каталоге, сначала по каталогу определяется индекс файла, затем отыскивается его индексный дескриптор в файловой системе, проверяются содержащиеся в нем права доступа и, наконец, отыскивается начало данных.

Следует также иметь в виду, что UNIX рассматривает устройства — терминалы, магнитные ленты, диски и даже оперативную память — как специальные файлы, имеющие, следовательно, индексные дескрипторы. Эквивалентность файлов и устройств глубоко заложена в UNIX, поэтому программы могут принимать ввод как из файлов, так и из устройств, в зависимости от того, какую информацию вы задали интерпретатору командного языка.

9.6.2. МОНТИРУЕМЫЕ ФАЙЛОВЫЕ СИСТЕМЫ

Одна из файловых систем отмечается в любой конфигурации как главная файловая система. Главная файловая система содержит существенную информацию, необходимую для запуска UNIX. Когда UNIX загружается и работает в однопользовательском режиме, вы работаете именно с главной файловой системой. Как же работают другие файловые системы?

Они располагаются на других дисковых устройствах и первоначально не известны UNIX. Способ подключения таких файловых систем к UNIX состоит в «монтировании» файловой системы к главной. Допустим, мы хотим монтировать файловую систему на диске /dev/rp9. Сначала нужно найти место для монтирования — это может быть простой каталог. Пусть у нас есть созданный специально для целей монтирования пустой каталог с именем /fs. Для монтирования в него файловой системы на диске /dev/rp9 можно дать команду:

```
польз. →  /etc/mount /dev/rp9 /fs<r>
```

После такого монтирования первоначальный каталог /ts и все его содержимое недоступны, пока /dev/rp9 не демонтирована или система не остановлена. Любая ссылка на /fs теперь будет ссылкой на самый верхний каталог файловой системы диска /dev/rp9. Если бы мы монтировали /dev/rp9 в файл /fstwo, то файл (на монтируемой системе), который теперь называется /fs/dir1/file1, назывался бы /fstwo/dir1/file1. Таким образом, фактический размер всей файловой системы UNIX, доступный обычному пользователю, может значительно превышать размер отдельного диска.

Отметим, что программа, обратная mount, называется umount (но не unmount):

```
польз. →  /etc/umount /dev/rp9<r>
```

(Отметим также, что обе эти программы находятся в каталоге /etc, а не в /bin.)

У вас, как правило, будет стандартный набор файловых систем, монтирующихся в стандартные места. Команды mount для этой цели должны располагаться в /etc/rc, который запускается всякий раз при переходе от однопользовательского режима к многопользовательскому. Тип конкретных файловых систем зависит от реализации. Обычно это одна главная файловая система и одна или несколько файловых систем пользователей. У вас может быть целая файловая система для временной памяти (/tmp). Но вы, если хотите, можете расположить временный каталог /tmp в главной файловой системе. Правда, это решение не лучшее, поскольку /tmp имеет тенденцию к сильному росту.

Вопросы о том, где должны находиться файловые системы и как велики они должны быть, решаются в зависимости от имеющегося оборудования и от характера использования системы.

9.7. СОХРАНЕНИЕ (ЗАЩИТА НА ЛЕНТЕ) ФАЙЛОВ

Наиболее важная задача администратора системы — регулярная защита (или обеспечение такой защиты) информации на ленте. Это означает, что некоторые или все файлы дисковых файловых систем должны записываться на магнитную ленту так, чтобы их можно было найти в случае утери файла. Причины утери следующие: случайное удаление файла пользователем; намеренное удаление файла пользователем, который затем изменил свое решение и желает восстановить файл; потери файла в результате системной ошибки. Возможна также ситуация, когда вы как администратор системы и привилегированный пользователь по ошибке удалите чей-то файл. Следует, конечно, быть очень внимательным, чтобы не допустить этого. Ошибаясь первый раз так «ужасно» (удалив чей-либо важный файл), вы в дальнейшем будете более осторожны. Если по одной из вышеуказанных причин файл все же удален, и у вас нет его последней копии на ленте, остается только сожалеть.

9.7.1. КОГДА ВЫПОЛНЯТЬ ЗАЩИТУ ФАЙЛОВ

Программа dump, описываемая ниже (более подробно см. в разделе 9.10), позволяет записывать на ленту только файлы, измененные в течение заданного промежутка времени. Если вы зададите специальное значение времени 0, то будет записываться вся файловая система.

Насколько сложной должна быть каждая процедура защиты файлов и когда нужно это делать, зависит от активности измене-

ния вашей файловой системы. Предположим, например, что полиая защита всей системы требует у вас три 800-метровых леиты, но каждый деиь меняется только иебольшая часть файловой системы. Вы могли бы делать полиую защиту каждую иеделю и так иазываемую «пошаговую» защиту иа магиитиую леиту ежедиевно. Тогда для восстаиовлеиия полиой системы, соответствующей любому дию недели, вам потребуется полиая иедельная лента и одиа пошаговая диевиая лента. Если ваша файловая система существеино меияется в течеиие каждого рабочего дия, имеет смысл производить ежедиевио полную защиту и брать только одиу леиту для каждой файловой системы. При устаиовке системы вы должиы подыскать разумиую схему защиты иiформации, одиако иеобходимо представлять себе все ее достоииства и иедостатки с тем, чтобы помеиять схему при изменеиии характера работы файловой системы.

Необходимо рассмотреть также вопрос о том, как долго следует храиить иiформацию иа леите. С одиой стороиы, количество леит ограиичеио и естествеиио использовать их миогократио. С другой стороиы, даже самая лучшая в мире процедура защиты файлов становится бесполезиой, если файлы уже были где-то записаны раиее. Одиа из эффективиых схем защиты состоит в следующем: сохраиять пошаговые ежедиевиые леиты до коица иедели, пока ие будет сделаиа иедельиая защита, сохраиять иедельиые леиты в течеиие месяца, сохраиять месячиые леиты всегда, причем вие машииного зала, чтобы иикто ие смог иа иих что-либо записать. Последиее предложеиие выглядит излишеством, но в одии прекрасиый деиь кто-иибудь придет к вам и попросит то, что было им выброшеио полгода иазад, и вы к его радости достаиете имеиио необходимую версию.

9.7.2. КАК ВЫПОЛНЯТЬ ЗАЩИТУ ФАЙЛОВ

Программы защиты и восстановлеиия файлов UNIX (dump и restor) подробно описываются в разделе 9.10, здесь же излагаются общие прииципы их использоваиия. Возможио, вы захотите иаписать комаидиые файлы для автоматизации процедуры защиты после того, как вам стаиет ясио, что и как следует для этого делать.

Перед выполиеиием защиты файлов иеобходимо сиачала запустить программы проверки целостиости файловой системы. Одиа процедура защиты иiформации должиа выполияться с особой тщательиостью: если во время выполиеиия защиты произойдет разрушеиие файловой системы, то будут утеряиы все результаты операции.

Для выполиеиия «пошаговой» защиты файловой системы можио использовать комаиду:

```
польз. →  dump 9 /dev/rrk0
```

Будут записаиы иа леиту все файлы (файловой системы

/dev/rrko), измененные с момента последней полной защиты файловой системы.

По возможности защиту файлов следует выполнять при небольшой активности пользователей в системе, а еще лучше — при полном отсутствии таковой. Это уменьшает вероятность модификации файлов в то самое время, когда они записываются на ленту. Такая ситуация обнаруживается иногда программой dump и вызывает выдачу сообщения «phase error» (ошибка фазы). К счастью, программы защиты и проверки файловой системы не требуют, чтобы последняя была монтирована во время их работы.

9.7.3. КАК ВОССТАНАВЛИВАТЬ ОТДЕЛЬНЫЕ ФАЙЛЫ

Рассмотрим процедуру восстановления файлов. Предположим, что файловая система записана на одну ленту и что нас интересует только один файл.

Установим ленту с последней версией файловой системы, возможно, содержащей версию файла, которую мы хотим восстановить. Выполним команду:

польз. → restor x FILE

заменяя FILE на имя требуемого файла и удаляя из него все префиксы, соответствующие точке монтирования. Например, если файл имеет полное имя /a/b/c, но /a — имя точки монтирования, используем только /b/c. Другими словами, имена файлов на ленте относятся к файловой системе, защищенной на этой ленте.

Если программа restor сообщит, что файла нет, ваша лента, наверное, содержит «пошаговую» защиту. В этом случае нужно взять ленту с полной защитой и начать все снова. В противном случае программа выдаст индекс файла и напечатает сообщение «mount the desired tape volume» (установите нужный том). Затем файл будет прочитан с ленты и записан в ваш текущий каталог под цифровым именем, равным десятичному значению первоначального индекса файла. Если вы пожелаете сохранить старое имя, файл следует переименовать.

9.7.4. ВОССТАНОВЛЕНИЕ ВСЕЙ ФАЙЛОВОЙ СИСТЕМЫ

Если файловая система полностью разрушена, вы можете прийти к выводу, что ее следует целиком восстановить с магнитной ленты. Это крайнее средство и вот как им следует пользоваться. Прежде всего убедитесь в том, что файловая система, которую вы восстанавливаете, не монтирована.

Если разрушена главная файловая система, единственный способ восстановить ее состоит в остановке машины и копировании эталонной версии UNIX. В противном случае введите команду:

польз. → /etc/umount DEVICE

Umount может сообщить вам «mount device busy» (устройство занято). Это означает, что кто-то использует его в данный момент, т. е. либо там находится чей-то текущий каталог, либо открыт расположенный там файл. Попросите всех выйти из данной файловой системы (не забудьте сделать это и сами!) и повторите команду umount. Разумеется, вам нечего беспокоиться о других файловых системах, если вы работаете в однопользовательском режиме, поскольку они не будут в этом случае монтированы. Как только вам удастся демонтировать файловую систему, выполните следующую команду:

```
польз. →  /ect/mkfs DEVICE SIZE
```

Здесь DEVICE — имя диска с файловой системой; SIZE — ее размер в блоках. Команда полностью и безвозвратно уничтожит старую файловую систему на диске, поэтому убедитесь в том, что выполняете команду правильно — в частности, убедитесь в правильности именования файловой системы! Неплохая мера предосторожности — сохранение файловой системы на ленте перед ее уничтожением на диске.

Установите ленту с полной защитой файловой системы и дайте команду:

```
польз. →  restor r DEVICE
```

При желании можно затем установить ленту с последней «пошаговой» защитой и выполнить процедуру восстановления также и с этой ленты.

9.8. РЕМОНТ ПОВРЕЖДЕННОЙ ФАЙЛОВОЙ СИСТЕМЫ

Теперь, когда мы знаем, из чего состоит файловая система, изучим, как исправлять ее в случае повреждения. Слово «повреждение» в данном контексте относится к некорректности управляющей информации в файловой системе и к неправильным данным, содержащимся в ее файлах. Оно редко будет означать собственно физическое повреждение устройства. Искусство ремонта файловой системы требует опыта и уверенности действий. Необходимо некоторое время, чтобы вы почувствовали себя спокойно, особенно, если вы будете вспоминать о последствиях определенных ошибок. Если вы собираетесь ремонтировать файловую систему и боитесь все перепутать (особенно если подозреваете большое количество сложных повреждений), запишите ее предварительно на магнитную ленту.

Прежде чем начать ремонт файловой системы, убедитесь, что никто другой не работает с ней. Только вы должны выполнять какие-либо операции над системой. Наилучший способ достичь этого — перевод UNIX в однопользовательский режим.

9.8.1. ОСНОВНЫЕ ПРОГРАММЫ ПРОВЕРКИ

Как уже было упомянуто ранее, основные программы проверки целостности файловой системы следует запускать 1) когда система загружается, 2) когда система начинает вести себя непонятным образом и 3) ежедневно для профилактики. Нужно иметь в виду, что проверка «активной» файловой системы во время работы UNIX может привести к выдаче сообщений о несуществующих ошибках, поскольку во время проверок диск может находиться в процессе модификации. Более того, файловая система должна быть по возможности демонтирована во время исправления повреждений, в противном случае это будет похоже на выполнение операции на прыгающем пациенте.

Имеются две проверочные программы: icheck и dcheck. Программа icheck проверяет непротиворечивость структуры индексных дескрипторов и списка свободных блоков файловой системы. Программа dcheck проверяет непротиворечивость структуры каталогов. Вероятнее всего, у вас будет командный файл, называемый примерно chk, запускающий эти программы для всех ваших файловых систем. Отметим также, что для каждого блокориентированного устройства, такого, как диск или лента, в UNIX имеется соответствующее байториентированное устройство. Так, в дополнение к /dev/rk0 имеется /dev/rrk0 (то же, что и обычное устройство, но система выполняет операции ввода-вывода над ним гораздо быстрее). Программы icheck и dcheck могут и должны работать с этими устройствами, если желательно повысить скорость проверки.

Прежде чем идти дальше, отметим следующее. Мы будем демонстрировать способы исправления файловой системы, предполагающие изменение данных на диске. После того, как вы решили, что файловая система исправлена, еще раз для гарантии запустите программы icheck и dcheck. Предлагаемые здесь методы могут иногда давать побочный эффект, за которым нужно следить. Ремонт файловой системы, следовательно, процесс итеративный. Опыт поможет вам сократить число итераций, но запуск проверочных программ по окончании ремонта — хорошая практика, даже если вы пришли к выводу, что устранили все повреждения.

Если вы ремонтируете главную файловую систему, то должны остановить машину и перезагрузить систему сразу после устранения всех ошибок, поскольку UNIX хранит управляющую информацию о главной файловой системе как на диске, так и в памяти, перезаписывая ее периодически из памяти на диск. Ремонт файловой системы производится только на диске и, если вы позволите UNIX продолжать работу, неверные данные из памяти снова будут переписаны на диск, и ваши труды пропадут даром.

Основная трудность восстановления главной файловой системы в том, что ее нельзя демонтировать. Существуют два возмож-

ных решеиия — остаиовка машииы и перезагрузка, как описаио выше, или копироваиие эталоиной версии UNIX. Для реализации первого решеиия абсолютно иеобходим одиопользовательский режим, для второго рекомендуется иикогда не добавлять и не изменять файлы в главной файловой системе.

9.8.2. ПРОГРАММА icheck

Номер всякого блока из области блоков даииых исправной файловой системы принадлежит либо списку свободиых блоков, либо списку блоков данных некоторого файла, ио не двум этим спискам одновремеиио. Каждый блок (его иомер) должеи встретиться точио один раз в одиом из таких списков. Программа icheck определяет отклонения от этого правила. Номер блока, встретившийся более одиого раза, иазывается дублированиым. Номер блока, не встретившийся ни разу, иазывается пропущенным. Предполагается, конечно, что все номера блоков в списках лежат в правильном диапазоне (от 2+isize до fsize — 1). Нарушения этого правила также определяются программой icheck. Номера блоков вне указаииого диапазоиа считаются неверными.

Раиее мы дали «нормальиый» вывод программы icheck. Рассмотрим его сиова, на этот раз более детально:

```
/dev/rp1:
files 540 (r=446, d=47, b=13, c=34)
used 8751 (i=212, ii=7, iii=0, d=8525)
free 6614
missing 0
```

Строка files (файлов) содержит общее число всех типов файлов: r озиачает число обычиых файлов, d — число каталогов, b — блокориеитироваиных специальиых файлов, c — байториеитироваиных специальиых файлов. Строка used (использоваио) состоит из общего числа занятых блоков и косвенных блоков первой (i), второй (ii) и третьей (iii) степеии, а также числа блоков даииых. Строка free (свободиых) содержит число блоков, не используемых в настоящее время.

1. *Ошибки в списке свободных блоков.* Список свободных блоков — это список иомеров блоков, которые могут быть распределеиы системой UNIX для использования в качестве блоков даниых. Если список оказался разрушен, ситуации могут быть разные. (Пропущеииые блоки не доступны для пользователя.) Если пропущен один блок, — это не очеиь страшно (хотя блок, конечио, для нас будет потеряи). Если пропущеииых блоков сотии, — это уже проблема. С другой стороиы, когда блоки присутствуют в списке свободиых блоков, ио на самом деле их там быть ие должио, ситуация потенциально весьма опасна: блок, прииадлежащий другому файлу, система может распределить для нового файла, что прозит хаосом.

Программа icheck выдает три типа диагностических сообщеиий, соответствующих вышеуказаниым случаям.

missing
#dup; inode=0, class=free
#dups in free

Символ # заменяется в каждом случае соответствующим номером блока. Первый случай — довольно безобидная ситуация с утерянными блоками. В двух других случаях есть потенциальная опасность при условии активности файловой системы. Если появляется одна из этих ошибок, игнорируйте на время другие ошибки и выполните следующую восстановительную процедуру:

польз. → icheck —s /dev/???

Флаг — s означает, что заново будет построен список блоков, не входящих ни в какие файлы. Если это главная файловая система, остановите немедленно машину и перезагрузите систему. Ошибки иного типа обнаруживаются при выполнении процедуры «icheck —s».

2. *Ошибки в индексных дескрипторах.* Эти ошибки выявляются сообщениями:

b#bad; inode=i#; class=[iclass]
b#dup; inode=i#; class=[iclass]

Здесь iclass может быть косвенный блок 1, 2 или 3-й степени либо характеристика файла: small (маленький), large (большой), huge (очень большой), garg (сверхбольшой). В каждом из этих случаев (как bad, так и dup) вам придется уничтожить файлы, содержащие указанные блоки. В случае dup в сообщении не указываются все индексы дублированного блока, поскольку дублирование фиксируется только при втором появлении указанного блока в файле; первое появление блока также нужно отыскать для того, чтобы проверить его правильность. Для поиска всех файлов, связанных с дублированным блоком, следует выполнить команду:

польз. → icheck —b # /dev/???

Команда выведет список всех индексов файлов, связанных с блоком номер #. Все эти файлы следует удалить.

Замечание. Если появляется одно из этих сообщений с class= =huge или garg, вместе с ним может появиться некоторое количество других, побочных сообщений об ошибках, так как в неправильном косвенном блоке содержится большое количество управляющей информации, в том числе и неверной. Но не предавайтесь панике. Просмотрите все сообщения об ошибках (может быть, сотни) и найдите одно из них с неверным блоком и с class=data (large). Уничтожьте файл с индексом, указанным в этом сообщении, после чего весьма вероятно, что большинство сообщений об ошибках пропадет.

Поскольку дублированные блоки могут присутствовать как в списке свободных блоков, так и в файлах, при получении сообщения «dup» запустите «icheck —s». Если вам повезет, дублиро-

ванные блоки исчезнут. В противном случае необходимо удалять файлы, как описано выше.

Необходимо устранить все ошибки, связанные с icheck, прежде, чем приступать к процедуре dcheck.

9.8.3. ПРОГРАММА dcheck

Напомним, что каждый индексный дескриптор содержит счетчик связей, т. е. число записей в каталогах, ссылающихся на данный индекс. Как только последняя запись удалена, ядро операционной системы предпринимает шаги для фактического удаления файла: освобождает его индекс и все блоки с данными. Таким образом, ни один файл не может иметь нулевое число связей. Программа dcheck проверяет нарушения этого правила. В случае отсутствия ошибок dcheck будет просто печатать имя устройства, содержащего проверяемую файловую систему. Если ошибки обнаружены, программа выводит таблицу со следующими колонками:

inode	entries	linkount
#	#	#

Здесь inode — конечно, номер индексного дескриптора (индекс файла); entries — число вхождений, т. е. записей в каталогах, где данный номер (индекс) отмечен как файл, содержащийся в том каталоге; link count — число связей, указанное в индексном дескрипторе. В нормальной ситуации число вхождений в каталогах совпадает с числом связей каждого файла и при этом не равно нулю. В таблице отмечаются только те случаи, когда эти два числа не равны одно другому либо оба равны нулю. Посмотрим, что следует предпринять в обоих случаях.

1. *Число вхождений=число связей=0.* Индексный дескриптор выделен для файла, но не имеет связей с каталогами. Такая ситуация не опасна и устраняется путем очистки индексного дескриптора:

```
польз. →  clri /dev/??? inumber
```

Здесь используется соответствующее имя устройства; inumber — номер индексного дескриптора. Можно применять байториентированные устройства, кроме того, можно задавать несколько номеров, перечисляя их в одной команде, например:

```
польз. →  clri /dev/rhp00 241 1001
```

Если команда clri выполняется для байториентированного устройства, то после нее нужно выполнить команду sync. Будьте внимательны при вводе номеров индексных дескрипторов.

2. *Вхождений меньше, чем связей.* Если число вхождений равно 0, можно выполнить команду clri, как и в предыдущем случае. Если оба числа (число вхождений и число связей) больше нуля, остается только ждать лучших времен: по мере того, как файлы

будут удаляться из каталогов, число вхождений упадет до нуля и можно будет выполнить команду clri.

3. *Вхождений больше, чем связей.* Очень опасная ситуация. Файл входит в каталоги, но индексный дескриптор предполагает таких вхождений меньше, чем на самом деле. Когда счетчик связей упадет до нуля по мере удаления файлов, индексный дескриптор и файл будут освобождены и затем перераспределены. Но некоторые каталоги будут считать, что они еще содержат этот файл. В результате возникнет невообразимый хаос. Чтобы не допустить его, вы должны уничтожить файл и удалить его из всех каталогов. Эта процедура описывается в следующем разделе.

После исправления всех связанных с dcheck ошибок вернитесь назад к процедуре icheck для сбора пропущенных блоков, которые появятся в результате выполнения команд clri.

9.8.4. УНИЧТОЖЕНИЕ ФАЙЛА

Из этого раздела вы узнаете, как уничтожать файлы. В обычных случаях файлы удаляются командой rm, но в случае неисправной файловой системы команда rm может вызвать еще большие повреждения системы. Мы предлагаем более «аккуратный» метод, включающий освобождение индексного дескриптора и удаление ссылок на файл во всех каталогах. Предполагается, что мы знаем индекс файла. Он может быть получен по имени файла с помощью команды «ls —i» для каталога, содержащего данный файл. Если файл, который вы собираетесь удалять, является каталогом, прочитайте следующий раздел этой главы.

Перед тем, как уничтожить файл, вы, возможно, захотите попытаться восстановить его содержимое. Имя файла можно получить при помощи флага —i команды ncheck. Эта команда, вероятно, подскажет вам, насколько важен данный файл. Наилучший способ сохранения файла перед его уничтожением состоит в простом копировании его (программой cp) в другую, исправную файловую систему. Создание нового файла в поврежденной файловой системе может только ухудшить ситуацию.

Разумеется, если в файле есть неправильные или дублированные блоки, его содержимое сомнительно. В случае дублирования блока в двух файлах один из файлов будет правильным, другой — нет.

Процедура уничтожения обычного файла (не каталога) выглядит следующим образом:

1) определите по номеру индексного дескриптора (индексу) имя файла, если вы его еще не знаете, для чего выполните команду:

```
польз. →  ncheck —i [inode#] /dev/???
```

где inode# — индекс файла, и вы получите список имен, связанных с данным файлом;

2) монтируйте файловую систему:

польз. → /etc/mount /dev/??? filesystemname

3) скопируйте интересующие вас удаляемые файлы в исправную файловую систему или на магнитную ленту;

4) выполните «rm —f» для каждого имени, полученного из ncheck;

5) демонтируйте файловую систему;

6) выполните clri для каждого индекса, который следует удалить из файловой системы;

7) сделайте sync;

8) снова выполните «icheck —s».

В случае ремонта главной файловой системы не забудьте перезагрузить систему!

9.8.5. УНИЧТОЖЕНИЕ КАТАЛОГА

Нельзя просто очищать индексный дескриптор повреждённого каталога, поскольку все содержащиеся в нем файлы будут потеряны.. Перед уничтожением каталога попытайтесь создать новый каталог где-либо в другом месте файловой системы и связать его со всеми файлами уничтожаемого каталога. Затем вы можете выполнить команды rm для всех имен файлов уничтожаемого каталога. Когда каталог пуст, его можно уничтожить, как это описано в предыдущем разделе (за исключением того, что вместо «rm —f» следует использовать rmdir).

Если повреждение настолько велико, что вы не можете найти ни одного файла в удаляемом каталоге, следует его уничтожить и снова выполнить процедуру dcheck. В этом случае вы наверняка потеряете все содержащиеся в нем файлы и еще раз почувствуете значение регулярной защиты дисков на магнитной ленте.

9.9. ЗАЩИТНАЯ КОПИЯ UNIX

Хранение защитной копии операционной системы оказывается весьма полезным в тех случаях, когда главная файловая система выходит из строя. Если, например, вы случайно потеряете Shell или программу init, из-за ошибки оборудования или в результате ввода неправильной команды в привилегированном режиме вы не будете в состоянии воспользоваться системой вообще. При этом наличие защитной копии существенно сэкономит время, которое в противном случае потребуется для переноса UNIX с другой установки и последующей перестройки системы.

9.9.1. ЧТО ТАКОЕ ЗАЩИТНАЯ КОПИЯ UNIX

Защитная копия UNIX (или просто копия системы) — это копия главной файловой системы, хранящаяся на диске или ленте, обычно не устанавливающихся на машину. Если копия системы

находится вне машины, ни один сбой или повреждение машины не может испортнть ее.

Удобнее всего хранить копию системы на диске. Однако, если у вас только одно дисковое устройство, такой копии во многих случаях недостаточно и вам придется хранить ее на магнитной ленте. Использование копии системы на ленте похоже на процесс первоначальной постановки UNIX на вашу машину.

9.9.2. РАЗЛИЧНЫЕ КОНФИГУРАЦИИ ЯДРА СИСТЕМЫ

В целях безопасностн зачастую полезно держать одну или более различных конфигураций ядра системы в дополнение к уже имеющейся при обычной загрузке. В этих случаях каждую версию ядра следует хранить как в главной файловой системе, так и в защнтной копии системы.

Например, автор во время написания данной книгн работал на машине с одним большим диском и четырьмя малыми. У рабочей конфигурации ядра системы (файл с именем /unix) была главная файловая система и область свопннга на большом диске. Альтернатнвные же версии (rkunix) были построены в расчете на работу с малыми дисками. Мы загружали их тогда, когда не работал большой диск.

В качестве другого примера рассмотрим установку с двумя дисковыми устройствами, каждое из которых имеет съемный диск. Однн диск содержит 9000 блоков. Устройства снабжены переключателями, позволяющими менять номера устройств, что дает возможность загружать систему с любого диска. Установка требует, чтобы все 9000 блоков были заняты под главную файловую систему, таким образом область свопинга должна располагаться на другом диске. Конфигурация ядра с меньшей главной файловой системой используется при защите диска. Она содержит на одном диске как главную файловую систему, так и область свопинга. Это необходимо для того, чтобы при защите диска не перекрывать его областью свопинга.

9.9.3. РЕМОНТ ГЛАВНОЙ ФАЙЛОВОЙ СИСТЕМЫ

В случае небольшого повреждения файловой системы не обязательно пользоваться защитной копией, как это описано в разделе 9.9.1. Переведите систему в однопользовательский режим, выполните необходимые очистки clri, процедуру «icheck —s», затем остановите и перезагрузите снстему, не выполняя команды sync (так как она обычно выполняется перед остановкой процессора). Этот метод уже можно не сравнивать с операцией на прыгающем пациенте, скорее, это маленькая операцня на самом себе.

Однако, если требуется значительный ремонт рабочей главной файловой системы, лучше воспользоваться ее защитной копией. Тогда во время ремонта главной файловой системой будет ее за-

щитиая копия, а вашу поврежденную рабочую систему нужно демонтировать, как и при ремонте обычиых пользовательских файловых систем.

Не забудьте о том, что во время ремонта у вашей рабочей главной файловой системы уже другое имя. Предположим, например, что у вас есть два дисковых устройства RK05 и в рабочем состоянии главной системой является /dev/rk0, а /dev/rk1 моитируется в каталог /usr. Предположим, далее, что вы решили монтировать защитную копию для ремоита главиой системы. Разумными процедурами были бы: замена (с остановкой процессора) диска /usr защитиым диском, переключение номеров устройств и запрузка защитной копии системы. Диск, который обычно иазывался /dev/rk0, теперь стаиет /dev/rk1 .

Вышеизложенная процедура ремонта главной файловой системы пригодна также и в случае восстановления главной системы с магнитной ленты. Вернемся к примеру с дисками RK05 и предположим, что необходимо восстановить главную рабочую систему с ежедневной полной защитой ленты. Для этого мы устаиавливаем требуемую ленту и выполняем команды:

```
польз. →  /etc/mkfs /dev/rrk1 4000 <r>
польз. →  restor r /dev/rrk1 <r>
```

Перед переключением на dev/rrk1 <r> проверим его.

```
польз. →  icheck /dev/rrk1 <r>
```

После того как главная файловая система восстановлеиа с ленты и проверена с помощью команд «icheck» и «dcheck», дайте команду «sync», остановите процессор, замените диск с защитиой копией, переключите устройства в иормальное состояние и запрузите обычную систему.

9.9.4. ОШИБКА В ПРОГРАММНОМ ОБЕСПЕЧЕНИИ, ИЛИ НЕИСПРАВНОСТЬ АППАРАТУРЫ!

Предположим, что придя иа работу вы обнаружили свою систему UNIX в нерабочем состояиии. Она печатает иа консоли сообщение «panic» (паника); остановлен процессор или оказывается иевозможным войти в систему. Стаидартиые действия в этом случае заключаются в остаиовке процессора (если ои еще не остановлеи), перезагрузке системы в однопользовательский режим и в выполиении процедур проверки файловых систем. Но предположим, что система не загружается. Тогда нужно попробовать запрузить защитную копию системы. Если и оиа не будет работать, по-видимому, неисправна аппаратура машины, и вам следует вызвать обслуживающий персоиал для ее ремонта.

С другой стороны, если защитиая копия запружается, вы должны проверить свою рабочую файловую систему с помощью процедур icheck и dcheck. После выполнеиия процедур и устраиеиия ошибок убедитесь в том, что файлы /etc/init и /bin/sh соответствуют своим защитиым копиям. Например, если ваша главиая фай-

ловая система по отношению к защитной копии имеет имя rk1 (при условии, что процедуры icheck и dcheck показали ее правильность), необходимо выполнить следующие действия:

```
польз.  →   /etc/mount /dev/rk1 /mnt <r>
польз.  →   cmp /mnt/etc/init /etc/init <r>
польз.  →   cmp /mnt/bin/sh / bin/sh <r>
UNIX    →   /mnt/bin/sh /bin/sh differ: char 22 line 2
польз.  →   cp /bin/sh /mnt/bin/sh <r>
польз.  →   sync <r>
```

В данном примере init оказалась в порядке, но shell был испорчен. Администратор системы затем скопировал shell с защитного диска («/bin/sh») на рабочий диск («mnt/bin/sh»).

9.9.5. ВОПРОСЫ

1. Кто является привилегированным пользователем и каковы его возможности?
2. Опишите процедуру добавления нового пользователя в систему.
3. Как перед остановкой системы проинформировать об этом всех работающих пользователей?
4. Как получить информацию о всех работающих процессах (пользовательских и системных)?
5. Когда чаще всего возникают ошибки в файловой системе?
6. Как узнать, сколько свободной памяти осталось в заданной файловой системе?
7. Каким образом можно автоматически инициировать специальные процедуры при переходе в многопользовательский режим работы системы?
8. Как программно подключить (или отключить) терминал к UNIX?
9. Что такое индекс файла?
10. Какой способ рекомендуется при защите файлов на магнитную ленту?*
11. Как восстановить файлы с ленты защиты?
12. О каких двух типах ошибок сообщается процедурой icheck?
13. Какая процедура рекомендуется для проверки файловой системы?

9.10. КОМАНДЫ ОБСЛУЖИВАНИЯ И ЗАЩИТЫ СИСТЕМЫ

Обслуживание операционной системы играет большую роль в UNIX, поскольку без него не может нормально функционировать ни одна система. В первых девяти разделах гл. 9 описывались обязанности администратора системы, а также подчеркивалась его ответственность за совершаемые действия. Эти действия почти невозможны без соответствующих средств, необходимых

* На вопросы 10—13 автор ответов не приводит. — *Примеч. пер.*

для каждодневного обслуживания системы. Система UNIX предоставляет такие средства, как запуск и остановка системы, добавление и удаление пользователей, запись файлов на ленту, восстановление их в случае разрушения и т. д. В первой половине гл. 9 описывались команды (набор средств), необходимые для выполнения указанных задач. В последующих разделах мы дадим несколько более подробную информацию о каждой команде.

Описываемые до сих пор методы обслуживания UNIX требуют большинства команд, рассматриваемых в этом разделе. Фактически эти команды объединяются в некоторые стандартные процедуры. Рекомендуется прочитать сначала первую часть гл. 9 и обращаться к данному разделу после того, как вы почувствуете необходимость в более точной информации о конкретных командах.

Описываемые далее команды применяются администратором при обслуживании операционной системы.

9.10.1. ПРОВЕРКА КОРРЕКТНОСТИ КАТАЛОГОВ ФАЙЛОВОЙ СИСТЕМЫ

Команда: dcheck
Синтаксис: dcheck [индексы] [файловая-система]
Действие: эта команда необходима для сравнения счетчика числа связей в индексном дескрипторе с числом записей в каталогах, ссылающихся на данный индексный дескриптор. Она также печатает имена файлов для заданных индексов и используется администратором системы (см. предыдущие разделы гл. 9).
Флаги: нет

Пример.

Dcheck запускается администратором системы для проверки целостности файловой системы. В предыдущих разделах гл. 9 более подробно описывался порядок выполнения этой процедуры. Приведем примеры выдачи информации командой dcheck как в случае наличия, так и в случае отсутствия ошибок.

```
польз. →  dcheck /dev/rp1 <r>
UNIX  →  /dev/rp1:
                    entries   link cnt    (число вхождений, счетчик числа
          →  448      0         0        связей)
          →  450      0         0
          →  653      0         0
          →  733      0         0
          →  1317     0         0
```

В данном примере все напечатанные числа вхождений в каталогах равны числам связей и все они равны нулю. Это говорит о том, что индексный дескриптор «занят», хотя и не имеет связей. Ситуация не опасна, и ее легко можно избежать, если очистить индексные дескрипторы командой clri.

```
польз.  →  dcheck /dev/rp1 <r>
UNIX    →  /dev/rp1:
                  entries     link cnt     (число вхождений; счетчик числа
           →  6       2           4        связей)
           →  39      1           0
           →  40      1           0
           →  46      0           0
           →  48      0           0
           →  51      0           0
           →  64      3           2
```

Здесь индексы 39 и 40 имеют вхождение в каталоге, ссылающееся на несуществующий файл. Файл с индексом 64 еще содержит данные (правда, сомнительной корректности). Все ссылки на эти индексы должны быть удалены, но сначала нужно попытаться сохранить содержимое соответствующих файлов.

Выводы

Каждый индексный дескриптор содержит счетчик числа связей, предполагающий такое же число записей о нем в каталоге. Как только файл освобождается, индексный дескриптор и все блоки с данными тоже освобождаются. Таким образом, ни один файл не может иметь нулевое число связей. Программа dcheck проверяет правильность выполнения этих условий.

Если dcheck не обнаруживает ошибок, она возвращает управление UNIX без выдачи каких-либо сообщений.

9.10.2. ПРОВЕРКА РАСПРЕДЕЛЕНИЯ ПАМЯТИ В ФАЙЛОВОЙ СИСТЕМЕ

Команда: icheck

Синтаксис: icheck [—s] [—b номера] [файловая-система]

Действие: эта команда исследует файловую систему, проверяя правильность списков свободных и использованных блоков. Каждый блок из области данных исправной файловой системы должен принадлежать либо списку свободных блоков, либо списку блоков некоторого файла. Каждый блок должен встречаться только в одном из этих списков, причем только один раз. Блок, встречающийся в списках более одного раза, называется дублированным, не встречающийся ни одного раза — пропущенным. Блок с номером, выходящим за пределы допустимого диапазона, называется неверным. Обычный вывод команды содержит следующую информацию:

общее число файлов, число обычных файлов, каталогов, блокориентированных и байториентированных специальных файлов;

общее число использованных блоков, число косвенных блоков первой, второй и третьей степени, число блоков с данными;

число свободных блоков;
число пропущенных (не найденных ни в одном из списков) блоков.

Флаги: в данной команде имеются два флага:

—s заново строит список свободных блоков, игнорируя старый список. При выполнении команды файловая система должна быть демонтирована. После выполнения команды icheck на главной файловой системе следует немедленно перезагрузить UNIX. Перед ее остановкой не нужно делать sync.

—b подавляет обычный вывод команды. За флагом должны идти один или более номеров блоков. Команда выдает диагностические сообщения для заданных блоков при условии, что они действительно встречаются в файловой системе.

Пример.

Сначала рассмотрим вывод icheck, не содержащий сообщений об ошибках.

```
польз. → icheck /dev/rp1 <r>
       → /dev/rp1:
       → files      739 (r=619, d=73, b=13, c=34)
       → used     11571 (i=277, ii=10, iii=0, d=11274)
       → free      3797
       → missing      0
```

Вывод команды представляет собой следующее:

files: содержит общее число всех файлов, число обычных файлов (r), число каталогов (d), число блокориентированных специальных файлов (b), число байтоориентированных специальных файлов (c);

used: содержит общее число использованных блоков, число косвенных блоков первой (i), второй (ii), третьей (iii) степени, число блоков с данными (d);

free: содержит число свободных блоков;

missing: содержит число пропущенных блоков.

```
польз. → icheck /dev/rp1 <r>
       → /dev/rp1:
       → files   739     (r=619, d=73, b=13, c=34)
       → used  11571     (i=277, ii=10, iii=0, d=11274)
       → free 3
       → missing   3794
```

Из примера мы видим, что имеются блоки, пропущенные в обоих списках. Это одна из наиболее часто встречающихся ошибок. Ее легко устранить путем выполнения команды icheck с флагом —s, с последующей перезагрузкой системы (без sync). Если ситуацию не изменить, то работать будет нельзя из-за отсутствия свободных блоков.

Выводы.

Важной функцией icheck является подтверждение целостности файловой системы, поскольку работа с неисправной файловой системой потенциально грозит потерей большого количества ин-

формации. Мы привели пример с ошибкой в списке свободных блоков. Есть, однако, и более критические ситуации, когда в список свободных блоков попадают блоки с полезной информацией, что крайне нежелательно. Более детально эта проблема обсуждалась в разделе 9.8.

9.10.3. ГЕНЕРАЦИЯ ИМЕН ФАЙЛОВ ПО ЗАДАННЫМ ИНДЕКСАМ

Команда: ncheck

Синтаксис: ncheck [—i индексы] [—a] [—s] [файловая-система]

Действие: эта команда генерирует полные имена файлов для заданного списка индексов файловой системы. Ее основная функция состоит в поиске имен поврежденных файлов.

Флаги:

—i печатает список полных имен файлов для индексов, перечисленных после данного флага.

—a печатает тот же список, что и для флага —i и дополнительно все файлы, имена которых начинаются с «.» и «..» (которые не печатаются для флага —i).

—s печатаются только специальные файлы и файлы с установленным режимом смены идентификатора пользователя. Этот флаг применяется для обнаружения скрытых нарушений правил защиты информации.

Пример.

В результате запуска команды dcheck мы обнаружили ошибки, связанные с индексами 39, 40 и 64. Для выяснения того, к каким файлам и каталогам относятся эти индексы, мы даем команду ncheck:

```
польз.  →   ncheck —i 39 40 64 /dev/rp1 <r>
UNIX    →   /dev/rp1:
        →   64      /xxx3
        →   40      /dq/trb/x
        →   39      /dq/trb/x2
        →   64      /working/xxx1
        →   64      /working/xxx2
```

Теперь мы можем продолжить исследование данных файлов и либо предварительно сохранить их, либо сразу очистить индексные дескрипторы командой clri.

Выводы.

Команда ncheck обычно применяется в тех случаях, когда мы хотим уничтожить файлы (не каталоги) и должны узнать для этого их имена. Как видно из последнего примера, имена файлов определяются по заданным номерам индексных дескрипторов (индексам). В предыдущих разделах гл. 9 данная команда описывается совместно с другими командами.

9.10.4. ОЧИСТКА ИНДЕКСНЫХ ДЕСКРИПТОРОВ

Команда: clri
Синтаксис: clri файловая-система индексы...
Действие: основная цель команды — удаление файла, отсутствующего по неизвестной причине во всех каталогах. Иногда необходимо удалять индексный дескриптор и для файла, присутствующего в некотором каталоге. В этих случаях следует сначала удалить его из каталога командой rm, а затем уже очищать индексный дескриптор. Перед тем, как воспользоваться командой clri, прочтите предыдущие разделы гл. 9 об обязанностях администратора UNIX.
Флаги: нет.

Пример.

Перед удалением индексных дескрипторов необходимо определить состояние файловой системы, чтобы больше узнать о тех индексных дескрипторах, которые собираемся удалять. Этой цели служат команды icheck, dcheck, ncheck. После определения подлежащих удалению индексов можно, например, выполнить само удаление следующим образом:

```
польз. →  clri /dev/rmt1 39 40 64 <r>
UNIX →  #
```

После очистки индексных дескрипторов следует выполнить команды «sync» и «icheck —s». Помните, что после ремонта главной файловой системы надо сделать перезагрузку.

Выводы.

Причинами очистки индексных дескрипторов могут быть: 1) число вхождений = числу связей = 0, 2) число вхождений меньше числа связей, 3) число вхождений больше числа связей. Следует помнить об особой опасности ошибок при очистке индексных дескрипторов, особенно если она выполняется без предварительного сохранения файлов. Для выяснения деталей применения команды clri (впрочем, как и других команд этого раздела) рекомендуется обратиться к началу гл. 9.

9.10.5. СОЗДАНИЕ ФАЙЛОВОЙ СИСТЕМЫ

Команда: mkfs
Синтаксис: /etc/mkfs [файловая-система] [размер]
Действие: эта команда создает новую файловую систему на диске или части диска в соответствии с числом блоков, заданным аргументом «размер». Такая система может затем использоваться после присоединения ее к главной файловой системе с помощью команды mount. Команда mkfs уничтожает предыдущее содержимое данной области диска. Созда-

ние новой файловой системы при условии, что существующая файловая система неисправна и ее нельзя отремонтировать (или для этого требуется слишком много усилий), полностью разрушит содержимое текущей системы, и взамен вы получите пустую файловую систему. Команда mkfs всегда должна выполняться при инициализации всякой файловой системы.

Каждый новый диск, поступающий в ваше распоряжение, должен генерироваться с помощью этой команды прежде, чем он будет использован. Заметим также, что команда mkfs не может выполняться прямо из каталогов /bin и /usr/bin, так как mkfs — специальная команда. Ею управляет администратор системы, и он решает, когда следует, а когда не следует создавать новую файловую систему. Как правило, каталог /etc доступен для чтения и записи только привилегированному пользователю.

Флаги: нет

Пример.

Нужно создать файловую систему на новом диске, содержащем 15 000 блоков. Сначала выясним, как будет называться наша файловая система, для чего просмотрим каталог /dev. Если в нем нет имени для файловой системы, мы должны создать его (вместе с характеристиками) с помощью команды mknod или непосредственно командой mkfs со специальным прототипом.

В нашем примере будем считать, что имя файловой системы уже существует. Для создания новой файловой системы дадим команду:

```
польз. →   /etc/mkfs /dev/rp3 15000 <r>
UNIX →   $
```

Выводы.

Как уже было указано, в случае, если имя файловой системы уже есть в каталоге /dev, нужен простой формат команды, приведенный в предыдущем примере. Если имени нет, придется задавать значительно больше информации — либо непосредственно в команде mkfs, либо с помощью дополнительных команд.

9.10.6. СОЗДАНИЕ СПЕЦИАЛЬНЫХ ФАЙЛОВ

Команда: mknod

Синтаксис: /etc/mknod имя [c] [b] тип устройство

Действие: эта команда используется для построения специальных файлов. Они обычно располагаются в каталоге /dev, где описываются характеристики драйверов устройств (дисков, лент и т. п.) и файловых

систем, имеющихся в данной конфигурации UNIX. Номера «тип» и «устройство» относятся соответственно к собственно драйверу (например, тип — драйвер терминала tty) и к специальному входу в драйвер (устройство — конкретный терминал в драйвере tty). Эти номера имеют специальное значение в каждой конфигурации системы и, если на них нет документации, их смысл можно определить по исходным текстам системы в файле conf.c. Как и другие команды специального назначения, команда mknod расположена в каталоге /etc, поскольку она выполняет системные функции по созданию специальных файлов.

Флаги: нет.

Пример.

Предположим, что нам нужно создать новое имя для существующего устройства tty (терминала). Сначала мы должны найти номер типа и номер устройства при помощи команды «ls —l /dev», которая выдает список специальных файлов вместе с номерами типа и устройства, именами файла, а также указанием на то, является ли устройство байториентированным или блокориентированным. Выдача может выглядеть следующим образом:

```
total 1
crw-w-w-   1 bin 0,   0   Jan 20 17:40   console
c-w-w-w-   1 bin 0,   40  Jan 20 17:38   lp
brw-rw-rw- 1 bin 1,   0   Jan 15 16:40   mt0
crw-rw-rw- 1 bin 3,   0   Jan 20 17:26   rmt0
brw-r-----  1 bin 0,  1   Jan 16 18:01   rp1
brw-r-----  1 bin 0,  2   Oct 29 04:07   rp2
brw-r-----  1 bin 0,  3   Dec 16 10:40   rp3
crw-r-----  1 bin 2,  1   Jan 16 18:03   rrp1
crw-r-----  1 bin 2,  2   Oct 29 04:07   rrp2
crw-r-----  1 bin 2,  3   Oct 29 04:07   rrp3
crw-w-w-   1 bin 0,   2   Jan 20 17:02   tty1
crw-w-w-   1 bin 0,   6   Jan 21 07:18   tty2
crw-w-w-   1 bin 0,   1   Jan 20 17:02   tty3
```

Мы видим, что первый символ каждой строки дает нам класс устройства (блокориентированное b или байториентированное c). Далее мы можем обнаружить номера типа и устройства (например, 0 и 1) и в конце строки — имя специального файла. Остальная информация та же, что и для обычных файлов.

Если нам нужно создать новую запись (имя) с характеристиками, подобными уже существующим записям, мы можем сделать это командой:

польз. → /etc/mknod tty0b c 0 1 <r>
UNIX → $

После чего можио посмотреть, что мы создали с помощью команды «ls —l/dev»:

```
crw-w-w-   1 bin 0,   1   Jan 20 17:40   console
c-w-w-w-   1 bin 0,  40   Jan 20 17:38   lp
brw-rw-rw- 1 bin 1,   0   Jan 15 16:40   mt0
crw-rw-rw- 1 bin 3,   0   Jan 20 17:26   rmt0
brw-r--    1 bin 0,   1   Jan 16 18:01   rp1
brw-r--    1 bin 0,   2   Oct 29 04:07   rp2
brw-r--    1 bin 0,   3   Dec 16 10:40   rp3
crw-r--    1 bin 2,   1   Jan 16 18:03   rrp1
crw-r--    1 bin 2,   2   Oct 29 04:07   rrp2
crw-r--    1 bin 2,   3   Oct 29 04:07   rrp3
crw-w-w-   1 bin 0,   1   Jan 21 14:34   tty0b
crw-w-w-   1 bin 0,   2   Jan 20 17:02   tty1
crw-w-w-   1 bin 0,   6   Jan 21 07:18   tty2
crw-w-w-   1 bin 0,   1   Jan 20 17:02   tty3
```

Мы только что создали новую запись tty0b с теми же характеристиками, что у tty3. Если мы пожелаем создать запись с характеристиками, отличающимися от существующих, то иам следует разобраться в ограничениях, накладываемых иа создание иовых специальных файлов. Такую информацию можно получить из исходиых текстов системы (файл conf.c).

Выводы.

Выполнение комаиды mknod прямо связаио с драйверами и предполагает существеииые знаиия внутреннего содержания системы гораздо большие, чем даются в книге. Ядро UNIX содержит около 10 000 строк исходного текста, из которых около 800 иаписаиы на языке Ассемблера, остальные — на языке Си.

9.10.7. МОНТИРОВАНИЕ ФАЙЛОВОЙ СИСТЕМЫ

Команда: mount
Синтаксис: /etc/mount файловая-система [—r] имя-файла
Действие: единственной файловой системой, которая автоматически монтируется при загрузке, является главная файловая система. Она содержит всю информацию, необходимую для запуска UNIX. Остальные файловые системы (см. команду mkfs) ие известны UNIX до тех пор, пока оии ие будут моитированы с помощью команды mount, либо явно, либо автоматически при переходе системы в многопользовательский режим работы (см. файл /etc/rc из гл. 9). Однажды монтированная файловая система

остается таковой до остаиовки системы или выполнения комаиды демонтироваиия.

Флаги: имеется одии флаг —r, означающий, что файловая система должиа моитироваться только для чтения.

Пример.

Монтироваиие файловой системы с именем rp3.

```
польз. →  /etc/mount /dev/rp3 programs <r>
UNIX →  $
```

Файловая система rp3 определена в каталоге /dev, который содержит специальиые файлы (драйверы устройств и т. п.). Если она там ие определеиа, система выдаст сообщение об ошибке. Имя programs должно быть каталогом, уже существующим в момент выполнеиия команды. После выполнеиия команды файловая система rp3 будет доступиа в главной файловой системе через каталог programs.

Выводы.

Если вы забудете моитировать файловую систему rp3 и попытаетесь получить доступ к расположениым иа ней даниым, обнаружится, что каталог programs трактуется как иекоторый другой каталог в главной файловой системе. Кроме того, требуется, чтобы каталог, являющийся корием каждой моитируемой файловой системы, был предварительно создаи в главиой файловой системе UNIX*.

9.10.8. ДЕМОНТИРОВАНИЕ ФАЙЛОВОЙ СИСТЕМЫ

Команда: umount
Синтаксис: /etc/umount файловая-система
Действие: эта комаида просто демоитирует заданную файловую систему, если последияя была монтироваиа. Если она ие была монтироваиа, выдается соответствующее сообщеиие.
Флаги: иет.

Пример.

Демоитируем файловую систему, монтированную в предыдущем примере.

```
польз. →  /etc/umount /dev/rp3 <r>
UNIX →  $
```

---

* Совсем ие обязательно. Файл, который после монтирования станоится корнем файловой системы, не обязан до этого быть каталогом и располагаться в главной файловой системе. Требуется только его существование. — *Примеч. ред.*

Выводы.

Обычно команда umount используется администратором системы. Так же как и другие специальные команды, она располагается в каталоге /etc и может быть выполнена только при наличии у пользователя доступа к этому каталогу.

9.10.9. ВРЕМЕННАЯ СМЕНА ИДЕНТИФИКАТОРА ПОЛЬЗОВАТЕЛЯ

Команда: su
Синтаксис: su [идентификатор]
Действие: эта команда позволяет сменить идентификатор пользователя и выполнить действия, которые, возможно, нельзя было бы выполнить с другим идентификатором пользователя из-за отсутствия у него прав доступа. Текущий каталог и командная среда пользователя остаются неизменными. Если новый идентификатор имеет пароль, система попросит вас его ввести. Для возврата к собственной среде необходимо нажать одновременно клавишу ctrl (управление) и букву d, т. е. ctrl/d. После этого вы окажитесь в той же командной среде, в какой были перед сменой идентификатора пользователя.
Флаги: нет.

Примеры.

(1) Чаще всего команда su используется при переходе в привилегированный режим, когда требуется доступ к каталогам и файлам, не доступным обычному пользователю. Для перехода в привилегированный режим нужно дать команду:

```
польз. →  su <r>
UNIX  →  Password:
польз. →                    введите пароль, эхо-отображение отсутствует
UNIX  →  #
```

С помощью приглашения «#» система сообщает о том, что вы стали привилегированным пользователем. Если вам не будет позволено стать привилегированным пользователем, вы получите приглашение «$».

(2) Можете ввести имя другого пользователя, например:

```
польз. →  su joe <r>
UNIX  →  Password:
польз. →                    введите пароль, эхо-отображение отсутствует
UNIX  →  $
```

Теперь можно войти в каталог другого пользователя с его правами.

Выводы.

Команда su необходима для получения доступа к защищенной информации. Ваш текущий каталог и командная среда при возврате в обычный режим будут восстановлены (возврат выполняется командой ctrl/d).

9.10.10. МОДИФИКАЦИЯ СУПЕРБЛОКА

Команда: sync
Синтаксис: sync
Действие: система UNIX обеспечивает всех своих пользователей буферизацией ввода-вывода. Когда файлы изменяются, удаляются, добавляются, действительная запись на диск происходит не сразу. Поэтому в случаях неожиданной остановки системы (нарушения питания, ошибки диска и т. п.) файловая система на диске не соответствует ее текущему состоянию, так как она может быть повреждена или даже разрушена. Посредством команды sync освобождаются буфера и модифицируется файловая система на диске. Sync автоматически выполняется системой через заданный промежуток времени (который устанавливается администратором системы и составляет обычно от 30 с до 2 мин). Каждый раз перед остановкой системы оператор должен давать sync, чтобы гарантировать соответствие памяти и диска.
Флаги: нет.

Пример.

Перед остановкой системы следует выполнить команду sync.

```
польз. →  sync <r>
UNIX  →  $
```

Выводы.

Единственное подтверждение того, что операция sync выполнена — появление на терминале приглашения UNIX. Если вы находитесь рядом с машиной, то можете в этот момент услышать движение головок дисков.

После остановки системы (при нарушении питания и т. п.) давать команду sync поздно. В таких случаях нужно приступить к проверке файловой системы с помощью описанных ранее команд (icheck, dcheck и др.).

9.10.11. БИБЛИОТЕКАРЬ МАГНИТНОЙ ЛЕНТЫ

Команда: tar
Синтаксис: tar [флаги] [имя]

Действие: эта команда сохраняет и восстанавливает файлы на магнитной ленте. Конкретные функции команды определяются флагами. Есть возможность сохранять и восстанавливать отдельные файлы и целые каталоги.

Флаги: делятся на две группы. Первая задает выполнение некоторой функции, вторая содержит модификаторы, добавляемые к заданной функции.

Допускаются следующие функции:

c создает новую ленту для записи на нее файлов (включает флаг «r»);

r заданные файлы записываются в конец ленты;

x заданные файлы читаются с ленты. Если на ленту записан целый каталог вместе со своими файлами и подкаталогами, флаг «x» вызывает чтение всего каталога. Если на ленте записано несколько версий данного файла, читается последняя версия;

t эта функция печатает список присутствующих на ленте файлов и каталогов, имена которых заданы в команде. Если не задано никаких имен, печатается полный список всех файлов ленты;

u заданные файлы добавляются на ленту, если их там еще нет или если они были изменены с момента их записи на ленту. Следующие флаги используются в качестве модификаторов вышеуказанных функций;

0,...,7 задает номер устройства, на котором установлена лента; по умолчанию 1;

v печатает имена всех файлов и каталогов, обрабатываемых заданной функцией. Вместе с флагом «t» выдает дополнительную информацию (помимо имен файлов);

w печатает название требуемого действия и имя файла, после чего система ждет ответа пользователя. Если вы отвечаете «y», действие выполняется, в противном случае — нет;

f следующий за данным флагом аргумент рассматривается как имя устройства вместо принятого по умолчанию имени /dev/mt?. Если имя задается дефисом («—»), tar будет читать стандартный ввод и писать в стандартный вывод. Таким образом, tar может быть использован в качестве фильтра;

b служит коэффициентом блокирования при чтении и записи файлов. По умолчанию принимается равным 1, максимальное значение равно 20;

l необходим для выдачи сообщения в случае, если при записи на ленту не удалось получить доступ ко всем файлам;

m сообщает программе tar, что не следует изменять время модификации при записи файлов на ленту. Время модификации файла остается равным времени последнего чтения файла с ленты.

Примеры.

(1) Сначала попытаемся создать на ленте архив файлов, принадлежащих одному каталогу. Для этого нужно либо перейти в данный каталог, либо задать полное имя файла.

```
польз.  →   tar c0 directory1 <r>
UNIX    →   $
```

tar запишет на ленту все файлы (и подкаталоги) каталога directory1. Все предыдущие данные на ленте будут разрушены. Если мы хотим проверять, что записывается на ленту, воспользуемся флагом «v», т. е. выполним команду:

```
польз.  →   tar cv0 work<r>
UNIX    →   a work/file1, 1 block
        →   a work/file2, 3 blocks
        →   a work/file3, 14 blocks
        →   a work/file4, 2 blocks
        →   $
```

Единственное различие этих двух случаев в том, что во втором мы затребовали печать имен файлов, записываемых на ленту. Буква «a» указывает на то, что файлы были добавлены на ленту, а число блоков означает, сколько блоков на ленте занимает данный файл. Как было указано ранее, флаг «c» инициализирует новую ленту, разрушая все ее предыдущее содержимое. Если бы мы захотели добавить новые файлы на ленту или заменить уже существующие, то следовало бы вместо флага «c» использовать флаг «r». В этом случае, если файл отсутствовал на ленте, он будет добавлен, если присутствовал, будет заменен.

(2) Теперь посмотрим, как можно блокировать файлы. Такой способ записи применяется, если нужно создать архив с большим количеством данных. Блокирование позволяет записать на ленту больше информации, чем в обычном случае, за счет уменьшения числа межблочных промежутков.

```
польз.  →   tar cv0b 16 work <r>
UNIX    →   a work/file1, 1 block
        →   a work/file2, 3 blocks
        →   a work/file3, 14 blocks
        →   a work/file4, 2 blocks
        →   $
```

Результат, выведенный на печать, не отличается от предыдущего, однако на ленте в данном случае может быть записано гораздо больше информации. Если теперь вы попытаетесь прочитать данную ленту без указания коэффициента блокирования, будет получено сообщение об ошибке. Вам придется повторить чтение с требуемым блокированием данных.

(3) Узнать содержимое некоторой ленты перед выполнением с ней каких-либо действий можно с помощью флага «t».

```
польз.  →   tar t0 work <r>
UNIX    →   work/file1
        →   work/file2
        →   work/file3
```

```
        →   work/file4
        →   $
```

Печатаются только имена файлов; счетчики байтов и число блоков не печатается.

(4) Теперь давайте прочитаем с ленты только что записанные нами файлы. Для этого следует использовать флаг «x». При чтении файлов мы должны находиться в каталоге, в который требуется поместить читаемые с ленты файлы.

```
польз.  →   tar xv0 work <r>
UNIX    →   x work/file1, 83 bytes, 1 block
        →   x work/file2, 34 bytes, 3 bloks
        →   x work/file3, 122 bytes, 14 blocks
        →   x work/file4, 40 bytes, 2 blocks
        →   $
```

В данном случае мы получили также размер каждого файла в блоках и дополнительно в байтах. Буква «x» указывает, что файлы были прочитаны.

(5) Теперь, когда мы научились создавать и читать файлы на ленте, посмотрим, как можно применять команду tar в качестве фильтра в начале или конце конвейера. Здесь посредством команды dd каждый файл из кода ASCII преобразуется в код EBCDIC. Сначала будем записывать на ленту, затем — читать с ленты.

```
польз.  →   tar cvf — file1 | dd of=/dev/rmt0 conv=ascii1 <r>
UNIX    →   a file1, 1 block
        →   $
```

Здесь с помощью программы tar создали новый архив на ленте, содержащий один файл file1, причем перед записью на ленту мы преобразовали его содержимое из кода EBCDIC в код ASCII.

Если мы затем пожелаем проделать обратную процедуру и прочитать файл с ленты, перекодируя его снова в EBCDIC, следует выполнить команду:

```
польз.  →   dd if=/dev/rmt0 conv=ebcdic | tar xvf — file1 <r>
UNIX    →   x file1, 1 block
        →   $
```

Команда dd рассматривается в разделе 7.2.2. Заметим, что, если команда tar выступает в качестве фильтра, необходимо задавать имя устройства /dev/rmt0 — «прозрачный» драйвер ввода-вывода с ленты mt0. Имена «прозрачных» устройств можно получить из каталога /dev.

Выводы.

Команда tar нужна всем пользователям; ее можно применять для защиты собственной информации на магнитной ленте.

9.10.12. СОХРАНЕНИЕ (ЗАЩИТА) ФАЙЛОВОЙ СИСТЕМЫ

Команда: dump

Синтаксис: dump [флаги [аргументы...] файловая-система]

Действие: эта команда нужна обычно администратору системы для обеспечения сохранности всех данных, содержащихся в файловой системе. Она предоставляет средства защиты файлов на основе таких критернев, как изменение файлов в течение дня и т. п., причем администратор системы может до минимума сократить время, требуемое для защиты информации. Ему не нужно выполнять ежедневно полную защиту всех файлов. Методы защиты информации обсуждаются в начальных разделах гл. 9.

Флаги: допустимы следующие флаги:

f — выполнить защиту на устройство, заданное следующим аргументом (вместо магнитной ленты по умолчанию);

u — при успешном завершенни защиты записать время начала этой процедуры в файл /etc/ddate. В нем отмечается время защнты каждой файловой системы, а также каждый уровень защиты;

0—9 — этот номер задает уровень защиты. Защите подвергаются все файлы, измененные с момента последней запнси в файл /etc/ddate для данной файловой системы с меньшими номерами уровней. Если уровень не задает никакого времени, предполагается начальное время. Такнм образом, флаг «0» вызывает защиту файловой системы целнком;

s — позволяет задать размер ленты (в футах)*. Следующий сразу за флагом «s» числовой аргумент задает размер ленты защиты. Когда при записи на ленту достигается заданная длина, программа останавливается и ждет смены ленты. По умолчанию размер ленты равен 2300 футов;

d — позволяет задать плотность записи информации на ленту, выраженную в битах на дюйм**. Аргумент идущий за данным флагом задает число требуемой плотности для вычнсления количества лент, необходимых при защите информации. Плотность записн по умолчанию подразумевается равной 1600 битов на дюйм (64 нмпульса на мм).

Пример.

Для выполнения пошаговой защиты файловой системы следует дать команду:

```
польз. →  dump 9 /dev/rrk1 <r>
UNIX →  #
```

---

* 1 фут — около 0,3 м. — *Примеч. пер.*

** 1 дюйм — около 2,54 см. — *Примеч. пер.*

Она запишет на ленту все файлы (файловой системы диска rrk1), измененные с момента последней защиты данной файловой системы.

Выводы.

Команда dump нужна администратору системы. Более подробно она рассматривается в гл. 9. Часто о команде dump необоснованно забывают до первого случая разрушения диска и потери всех его файлов. Не допускайте этого в вашей системе.

9.10.13. ВОССТАНОВЛЕНИЕ ФАЙЛОВОЙ СИСТЕМЫ

Команда: restor

Синтаксис: restor флаги [аргументы...]

Действие: команда служит для чтения магнитных лент, записанных командой dump.

Флаги:

f — позволяет использовать вместо принятой по умолчанию другую входную ленту. Имя ленты задается сразу вслед за данным флагом;

r или R — задает файловую систему, с которой лента должна читаться и загружаться. Имя файловой системы — аргумент, следующий за флагом r или R. Если вместо r берется R, программа restor запрашивает, какая из многотомных лент установлена для чтения. Это позволяет прерывать программу и запускать заново (перед каждым новым запуском следует выполнять «icheck —s»);

x — восстанавливаются файлы, указанные в качестве аргументов. Имена файлов должны быть без префиксов, обозначающих точку монтирования (т. е. файл /usr/bin/lpr на ленте имеет имя /bin/lpr). В предыдущих разделах гл. 9 рассматривается процедура минимизации числа чтений лент защиты;

t — печатает время записи ленты и время восстановления файловой системы.

Пример.

Чаще всего требуется восстанавливать отдельные файлы, разрушенные по некоторым причинам, для чего нужно ввести команду:

```
польз. →  restor x file1 <r>
UNIX  →  #
```

Помните, что имена файлов задаются без префиксов, соответствующих точке монтирования. Например, если файл имеет полное имя /usr/dick/file1 и /usr означает имя каталога для монтирования, следует использовать имя /dick/file1. Другими словами, имена файлов на ленте записываются относительно файловой системы, защищаемой на данной ленте.

Выводы.

Команда restor работает совместно с командой dump и, как правило, управляется администратором системы. Если при попытке восстановления файла оказывается, что его нет на ленте, то, по-видимому, вы установили ленту с пошаговой защитой. Тогда возьмите ленту с полной защитой и начните все сначала. Восстановление полной файловой системы с ленты защиты следует рассматривать как крайнее средство. О деталях восстановления файлов см. в предыдущих разделах гл. 9.

9.10.14. ВОПРОСЫ

1. Каково назначение следующих команд?
   а) icheck, б) ncheck, в) clri.
2. Для чего нужна команда mount?
3. Вы обнаружили, что не получаете доступа к файлу из-за отсутствия у вас прав чтения и записи. Что вы должны предпринять для получения доступа к файлу?
4. В каких случаях следует предпочесть команду dump команде tar?

Приложение А

СООБЩЕНИЯ ОБ ОШИБКАХ ЯДРА UNIX

Ядро UNIX печатает сообщения об ошибках на системной консоли в случае ошибок устройств ввода-вывода, переполнения таблиц системы, других ошибок операционной системы. Приводим сводку сообщений об ошибках.

err on dev d/d (ошибка на устройстве d/d)
bn ddddd er dddddd dddddd (блок dddddd регистры **dddddd** dddddd)

Это сообщение печатается в случае ошибки устройства ввода-вывода. Символы d/d обозначают номер типа и номер устройства, где произошла ошибка. Список номеров типов и номеров устройств и соответствующих им имен устройств можно получить следующей командой:

польз. → ls — l /dev | grep "^b" <r>

Вывод команды может иметь следующий вид:

brw-r----	1 bin	8, 0	Mar 13 15:13	hk00
brw-r-r-	1 bin	8, 1	Mar 27 12:06	hk01
brw-r----	1 bin	8, 2	Jan 13 14:32	hk02
brw-r----	1 bin	8, 3	Jan 13 14:32	hk03
brw-r----	1 bin	8, 4	Mar 13 15:18	hk04
brw-r----	1 bin	8, 8	Jan 13 14:32	hk10
brw-r----	1 bin	8, 9	Mar 10 09:30	hk11
brw-r----	1 bin	8, 10	Jan 13 14:32	hk12
brw-r----	1 bin	8, 11	Mar 13 23:22	hk13
brw-r----	1 bin	8, 12	Mar 10 16:53	hk14
brw-rw-rw-	1 bin	3, 0	Apr 10 17:45	mt0
brw-rw-rw-	1 bin	3, 4	Apr 10 01:27	mt4
brw-rw-rw-	1 bin	3, 32	Feb 12 21:44	nmt0
brw-rw-rw-	1 bin	3, 36	Aug 27 12:04	nmt4

Здесь перечислены «блокориентированные специальные файлы» каталога /dev, содержащего все специальные файлы, относящиеся к устройствам системы. «Блокориентированные специальные файлы» — дисковые и ленточные устройства.

Три первые колонки в действительности не отличаются от вывода команды «ls —l» для обычных файлов, за исключением того,

что буква «b» виачале указывает иа блокориентированный специальиый файл. Следующая колоика, содержащая для обычиых файлов размер файла в байтах, в даиием случае содержит разделеиные запятой иомер типа устройства и иомер устройства. Таким образом, сообщеиие об ошибке

er on dev 8/0 (ошибка на устройстве 8/0)
bn ddddd er dddddd dddddd (блок ddddd регистры ddddd ddddd)

указывает на ошибку устройства /dev/hk00, которое в даиием случае будет первой частью устройства 0 диска RK07.

Буквы ddddd после слова «блок» обозиачают номер блока, где произошла ошибка иа устройстве. Числа после слова «регистры» представляют собой содержимое регистров ошибок устройства. Какие это регистры и что оии содержат — зависит от типа устройства, где произошла ошибка.

bad block on d/d (неверный номер блока на устройстве d/d)

Это сообщение об ошибке печатается в случае повреждеиия файловой системы. Здесь d/d сиова номер типа и иомер устройства с повреждениой файловой системой. Ошибка означает, что некоторый файл затребовал блок, чей иомер находится за пределами файловой системы, которой прииадлежит даниый файл, и файловая система должиа быть подвергнута ремонту.

bad count on d/d (ошнбка в счетчнке блоков на устройстве d/d)

указывает иа повреждеиие файловой системы и означает, что счетчик числа свободиых блоков или свободиых иидексиых дескрипторов иаходится в ошибочиом состояиии. Снова, как и ранее, требуется ремонт.

no space on d/d (нет места на устройстве d/d)

указывает на то, что исчерпаиа свободиая область файловой системы иа задаином устройстве. Некоторые пользователи могут потерять свои файлы, если в процессе редактироваиия записывали их на диск, причем эти файлы иевозможио восстановить. Наилучший выход из положеиия — отправка с помощью /etc/wall всем пользователям сообщеиия о том, что даиной файловой системе иельзя создавать новые файлы и копировать на иее другие файлы, а также требование удалить все ненужные файлы из данной файловой системы.

out of inodes on d/d (много нндексных дескрнпторов на устройстве d/d)

озиачает, что исчерпаиа свободиая область в списке иидексиых дескрипторов файловой системы иа даиием устройстве. Все сказаиное о предыдущем сообщении «no space» отиосится и к этому случаю.

no file (много открытых файлов)

Программа пытается открыть файл, но в системной таблице открытых файлов нет места для очередного файла. Если ошибка будет периодически повторяться, рекомендуется с помощью /etc/wall предупредить об этом пользователей и увеличить размер таблицы открытых файлов.

inode table overflow (много индексных дескрипторов)

Программа пытается использовать файл, но в системной таблице активных индексных дескрипторов нет места для очередного файла. Так же как и в предыдущем случае, если ошибка будет периодически повторяться, следует предупредить пользователей и увеличить размер таблицы активных индексных дескрипторов.

panic: reason (крах системы: причина)

указывает на серьезную ошибку, препятствующую продолжению работы операционной системы. Машину следует остановить, отметив причину ошибки в системном журнале. Причины могут быть следующими:

«out of swap space» (переполнение области свопинга)

т. е. исчерпана область свопинга на диске, куда выгружаются программы из оперативной памяти. Следует либо отвести больше памяти для области свопинга, либо уменьшить максимально допустимое число процессов в системе.

«out of text» (много разделяемых процедурных сегментов)

Система пытается выполнить программу, с процедурным сегментом которой могут одновременно работать несколько пользователей, однако в системной таблице разделяемых сегментов нет места для очередной программы. Следует увеличить размер таблицы разделяемых сегментов в системе.

«swap error» (ошибка при свопинге)

указывает на ошибку ввода-вывода при выгрузке программы в область свопинга либо при загрузке ее обратно в оперативную память. Перед данным сообщением на системной консоли обычно печатается сообщение об ошибке ввода-вывода. Если такая ошибка появляется периодически, попросите инженера-электронщика проверить дисковое устройство, где расположена область свопинга.

«no clock» (нет таймера)

указывает на то, что UNIX не в состоянии включить часы, входящие в состав аппаратуры машины. Если такие часы действительно установлены в машине, это указывает на ошибку аппаратуры.

«parity» (ошибка четности)

указывает на ошибку четности памяти, т. е. аппаратуры машины.

«(other)» («остальные»)

Остальные причины серьезных ошибок этого класса кроются либо в программном обеспечении операционной системы, либо в аппаратуре машины.

Приложение Б

СВОДНЫЙ ПЕРЕЧЕНЬ КОМАНД UNIX

(Приводятся только основиые возможиостн команд.)

Библиотекарь
ar флаги [имя] библиотека [файл...] флаги:
d удалить файлы нз библиотекн;
r заменнть файлы в библиотеке;
t печать имеи файлов нз библиотеки;
x извлечь файлы из библнотеки;
v печать дополнительиой информации;
c создать библиотеку.

Калеидарь
cal [месяц]год

Коикатеиация файлов
cat [—u][файл...]
—u задает размер блока (по умолчаиию 512)

Смеиа группы файла
chgrp группа файл...

Устаиовка кода защиты файла
chmod код-защиты файл...
код-защиты:
4000 смеиа идеитификатора пользователя;
2000 смеиа идеитификатора группы;
0400 чтение для владельца;
0200 запись для владельца;
0100 выполиеиие для владельца;
0070 чтеиие, запись, выполненне для группы;
0007 чтение, запись, выполнение для прочих.

Смена владельца файла
chown имя файл...

Очистка иидексиого дескриптора
clri файловая-система иидекс...

Сравиеиие двух файлов
cmp [—l] [—s] файл-1 файл-2
—l печать полиой таблицы различнй;
—s установка кода возврата.

Построчиое сравиеиие файлов
comm [—[123]] файл-1 файл-2
123 иомера печатаемых колоиок.

Копироваиие файлов
cp файл-1 файл-2
cp файл... каталог

Печать и устаиовка времеии
date [yymmddhhmm[.ss]]

Проверка корректности каталогов
dcheck [—i иидекс...][файловая-система]
иидексы геиерируются командой icheck

Преобразоваиие файла
dd [аргумеиты]...
аргумеиты:
if=имя входной файл;
of=имя выходной файл;
bs=n размер блока;
skip=n пропуск n входных записей;
files=n копировать n файлов с леиты;
seek=n найти запнсь n;
count=n копировать n записей;

conv=
ascii преобразовать EBCDIC в ASCII;
ebcdic преобразовать ASCII в EBCDIC;
lcase преобразовать все буквы в строчные;
ucase преобразовать все буквы в прописные.

Свободная память на диске
df [файловая-система]

Различия между двумя файлами
diff [—efbh] файл-1 файл-2
—e генерация команд редактора;
—b игнорировать лишние пробелы;
—f краткий список различий;
—h быстрый поиск различий.

Различия между тремя файлами
diff3 [—ex3] файл-1 файл-2 файл-3
—e вывод для редактора;
—x3 различия только для файла-3.

Использование диска
du [—s] [—a] [имя...]
—s общее число блоков;
—a число блоков для каждого файла.

Сохранеиие файловой системы
dump [флаги[аргумент...] файловая-система]
флаги:
f задает устройство для защиты;
u запись времени защиты;
0—9 уровень защиты;
s размер ленты;
d плотность записи на ленте.

Вывод аргумеитов
echo [—n][аргумент]...
—n отмена перевода строки

Тип файла
file имя...

Поиск файлов
find имя... аргументы...,
аргументы:
—name имя файла;
—mtime n файлы изменены в течение n последних дней;
—print печать найденных имен файлов.

Поиск строк по шаблоиу
grep [флаг]... выражение [файл]
флаги:
—v печать строк без шаблона;
—n печать строк с номерами;
—y сопоставление строчных и прописных букв.

Проверка распределеиия памяти в файловой системе
icheck [—s] [—b блок...] [файловая система]
—s создание списка свободных блоков;
—b выдача сообщений для данных блоков.

Посылка сигнала
kill [сигнал] процесс
сигнал задается номером

Системная печать
lpr [флаги]... [файл]...
флаги:
—r удалить файл после печати;
—c копировать файл перед печатью.

Содержимое каталогов
ls [—флаги...]имя...
флаги:
l полная информация;
t сортировка по времени;
a вывод всех имен (.и..)
d информация о каталогах

Почта
mail [имя]...

Прием и отмеиа сообщеиий
mesg [n] [y]

Создаиие каталогов
mkdir имя...

Создание файловой системы
/etc/mkfs[файловая-система] [размер]

Создание специальиых файлов
/etc/mknod имя[c][b] тип устройство

—c байториентированный файл;
—b блокориентированный файл.

Монтирование файловой системы

/etc/mount [файловая-система [—r]]
—r только для чтения

Переименование файлов

mv файл-1 файл-2
mv каталог-1 каталог-2
mv файл-1 ... файл-n каталог

Генерация имен файлов по индексам

ncheck [—i индекс ...] [—a] [—s] [файловая-система]
—i печатает полные имена файлов;
—s печатает имена специальных файлов.

Понижение приоритета команды

nice [—число] команда [аргументы]
число не более 20

Восьмеричный дамп файла

od [—bcdox] файл [[+смещение [.] [b]]
c символьный;
d десятичный;
o восьмеричный;
x шестнадцатеричный.

Печать файлов

pr [аргумент] ... [файл] ...
аргументы:
—n печать в n колонок;
+n печать с n-й страницы;
—h задание заголовка;
—wn ширина страницы;
—ln длина страницы.

Состояние процессов

ps [флаги ...] [имя]
флаги:
a все терминальные процессы;
x все процессы;
l полная информация.

Текущий каталог

pwd

Таблица с содержанием библиотеки

ranlib [библиотека ...]

Восстановление файловой системы

restor флаг ... [аргументы ...]
флаги:
f имя ленты защиты;
r или R имя файловой системы;
x восстановление отдельных файлов;
t печать времени защиты;
y размер ленты защиты.

Удаление файлов

rm [флаги] файл...
флаги:
— b безусловное удаление;
— r удаление всех файлов и подкаталогов;
— i интерактивное удаление.

Удаление каталогов

rmdir каталог ...

Задержка выполнения команды

sleep время
время задается в секундах.

Сортировка

sort [—флаги ...] [+pos1 [—pos 2]] ... [—o имя]
[—T каталог] [имя] ...
флаги:
b игнорировать пробелы в начале строки;
f соответствие прописных и строчных букв;
m слияние;
n арифметическая сортировка;
o имя выходного файла;
u игнорировать одинаковые строчки.

Разбиение файла

split [—n] [файл [имя]]
— n число строк в выходных файлах.

Установка режима терминала

stty [аргументы ...]

Смена идентификатора пользователя

su [имя]

Модификация суперблока

sync

Установка табуляции

tabs [—n] [терминал]
— n левый отступ не выравнивается.

Библиотекарь магнитной ленты
tar [флаги] [имя]
флаги:
r запись файлов в конец ленты;
x чтение файлов с ленты;
t перечень содержимого ленты;
u запись файлов при условии их отсутствия на ленте;
0...7 номер устройства;
v дополнительная печать имен;
b коэффициент блокирования.

Дублирование стандартного вывода
tee [флаги] [файл] ...
флаги:
— i игнорировать прерывание;
— a добавление к файлу.

Получение имени терминала
tty

Демонтирование файловой системы
/etc/umount файловая-система

Вывод одинаковых строк файла
uniq [—флаги [+n] [—n]]
[вход] [выход]
флаги:
u вывод неповторяющихся строк;
d вывод повторяющихся строк.

Сообщение всем пользователям
/etc/wall

Подсчет числа слов
wc[—lwc] [файл...]
l число строк;
w число слов;
c число символов.

Информация о работающих пользователях
who [файл] [amI]

Передача сообщения
write пользователь [терминал]

Приложение В

Ответы на вопросы

К главе 2

1. Запросить его у администратора системы.
2. По вашему усмотрению, оно должно быть кратким и легко запоминаемым.
3. Пароль должен содержать не менее 6 символов. Он может быть короче, но тогда должен включать управляющие или специальные символы.
4. Попросите администратора системы удалить его.
5. Один или более пробелов.
6. Нажать клавишу «возврат каретки».
7. В общем случае отвечает символом приглашения «$».
8. «$».
9. Введите «#» сразу за неправильным символом.
10. Удалить команду с помощью символа «@ ».

К главе 3

1. а) получаете сообщение «? имя файла»;
   б) получаете сообщение «n», где n — размер файла в байтах.
2. Когда вызывается режим добавления, редактор позиционируется непосредственно после той строки, которая была текущей в момент вызова этого режима.
3. Вы должны перейти на новую строку и ввести символы.
4. а) *добавление*
   a ⟨r⟩
   • ⟨r⟩
   добавляет новый текст
   текст вводится после текущей позиции
   б) *вставка*
   i ⟨r⟩
   • ⟨r⟩
   вставляет новый текст
   текст вставляется перед текущей позицией
   в) *замена*
   c ⟨r⟩
   • ⟨r⟩
   заменяет существующие строки текстом

заменяется текст в текущей позиции
г) *удаление*
d ⟨r⟩
ничего не нужно делать
удаляет текущую строку текста
удаляется строка текста в текущей позиции
д) *печать*
p ⟨r⟩
ничего не нужно делать
печатает текущую строку текста
ничего не изменяется

5. Используется команда подстановки:

s/старый текст/новый текст/

6. Используется команда «w». Редактор ed отвечает числом записанных символов.

7. Команда «q».

8. Команда «1,$p».

9. Редактор выводит символ «?»

10. Символ «$» в зависимости от контекста означает либо конец строки, либо конец файла;

символ «^» означает начало строки текста;

символ «*» означает любое количество последовательных вхождений одного и того же символа;

символ «g» означает «глобальный» режим команды подстановки. Он идет в конце команды и означает подстановку всех вхождений в текущей строке или в каждой строке заданного диапазона.

11. а) печатает номер текущей строки;
б) печатает строку, предшествующую текущей;
в) печатает строку, следующую за текущей;
г) печатает текущую строку;
д) заменяет строки 1, 2, 3 одной строкой «new line of text»;
е) удаляет текущую, а также 3 следующих строки;
ж) находит первое появление текста «The»;
з) добавляет текст «the» в начало текущей строки;
и) записывает весь файл (от первой до последней строки) в новый файл с именем «newfile»;
к) добавляет символ «$» в конец текущей строки текста;
л) заменяет в текущей строке все вхождения текста «I» на текст «you».

К главе 4

1. В свой корневой каталог (т. е. если ваше имя «dick», то, скорее всего, в каталог «.../dick».)

2. Ограничения устанавливает администратор системы. Вы можете создавать файлы и подкаталоги до тех пор, пока UNIX не выдаст сообщения о переполнении файловой системы.

3. Иерархическая файловая система.

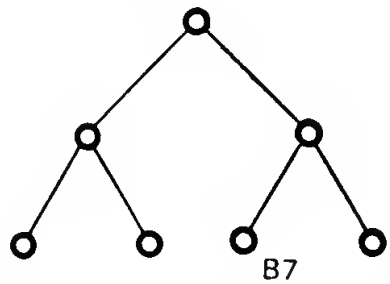
4. Дайте команду pwd.

5. ls перечисляет файлы и подкаталоги текущего каталога;
cd выполняет переход в другой каталог;
mkdir создает новые каталоги;
rmdir удаляет существующие каталоги;
rm удаляет файлы.

6. Дайте команду «ls —l». Все каталоги начинаются со строки «drw...», обычные файлы — со строки «—rw...».

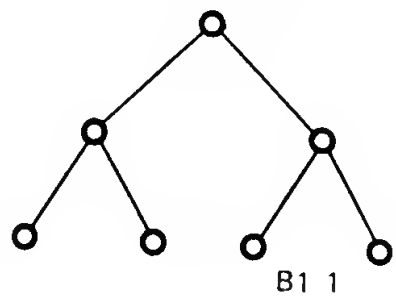
7. Владелец (тот, кто вошел в систему), группа, прочие пользователи.

8. а) установка в каталог dick;
б) установка в dick/A/A1;
в) установка в dick/A/A2;
г) установка в dick/B;
д) удаление каталога B1;



е) ошибка: «не удалось удалить каталог A» (неправильная установка текущего каталога);
ж) dick/B;
з) создание каталога B1.1

Новая файловая структура выглядит следующим образом:



К главе 5

1. Результат команд выглядит следующим образом:

а) 1 1 1 1 1 1
1 1 1 1 1 1
2 2 2 2 2 2
2 2 2 2 2 2
$

б) $ данные записаны в file3;

в) система будет ждать ввода с терминала и поместит данные в файл «file4».

2. Файл «file1» будет поставлен в очередь системной печати, после чего будет напечатан, как только освободится печатающее устройство. UNIX вернет управление пользователю с помощью приглашения «$», как только файл будет поставлен в очередь.

3. Команда pr предназначена для простейшего форматирования вывода текста с помощью нумерации страниц и выдачи заголовка на каждой странице. Вывод направляется не на устройство печати, а в стандартный файл вывода.

4. а) вход в систему и установка текущего каталога dick;
б) переход в каталог A;
в) копирование файла F1 в каталог B с тем же именем файла F1;
г) переход в каталог dick;
д) копирование файла F2 из каталога A в каталог B и присваивание ему там имени FA;
е) копирование всех файлов каталога A в каталог B с теми же именами. Содержимое каталога A не изменяется;
ж) все файлы в каталоге B переименовываются таким образом, что будут содержаться в каталоге C. Каталог B становится пустым;
з) переименование файла FA из каталога C в файл F1 каталога B.

5. Владельцу и членам группы может быть разрешен доступ к некоторому файлу (файлам). Для этого может понадобиться сменить имя владельца и группы у файла (файлов).

6. а) чтение/запись/выполнение для владельца, чтение/выполнение для всех остальных:

```
-rwxr-x-r-x
```

б) чтение/запись для владельца и группы, только чтение для всех остальных:

```
-rw-rw-r-
```

в) чтение/запись/выполнение только для владельца:

```
-rwx-
```

г) чтение/запись для владельца, чтение для группы и всех остальных:

```
-rw-r-r-
```

К главе 6

1. Используется стандартный ввод-вывод. Ввод принимается с терминала пользователя, вывод направляется обратно на терминал.

2. Результаты команды записываются в новый файл благодаря символу «>». Если файл уже существовал, его содержимое заменяется новыми данными. Сохранение данных существующего файла и присоединение новых данных к старым осуществляется посредством двух символов «>>».

3. Используются символы «<» или «<<».

4. Процесс — это действие машины по выполнению одной задачи.

5. Да; с помощью символа «&», помещаемого в конец командной строки.

6. Да; с помощью программного канала, обозначаемого символом вертикальной черты «|», которая ставится между командой, посылающей данные, и командой, принимающей эти данные.

7. Это метасимволы, используемые для создания шаблонов, помогающих выбирать требуемые имена файлов и каталогов. Символ «*» обозначает любое число произвольных символов; символ «?» обозначает один произвольный символ.

8. Создается шаблон, по которому будут выбираться все имена, начинающиеся с букв a, b, c, d, ..., z. Затем печатается список с полной информацией о файлах с выбранными именами.

К главе 7

Средства связи

1. Да; единственное условие — задавать правильные имена пользователей.

2. Да; почтовые сообщения, снабженные датой и временем отправки, будут сохраняться до тех пор, пока вы их не затребуете.

3. Всякий раз, когда вы входите в систему или даете команду mail и при этом для вас действительно имеется почта.

4. Да, это команда wall.

5. Сообщение посылается, как только вы нажимаете ctrl/d на вашем терминале, причем сообщение приходит немедленно после его отправки.

6. Вы можете послать сообщение пользователю, работающему в настоящий момент в системе, при условии, что он не запретил прием сообщений командой mesg.

Обработка файлов

1. dd if=/dev/mt0 conv=ascii
2. diff —e file1 file2
3. grep Syntax*
4. а) выводит число строк, число слов и число символов файла file1;
   б) разбивает файл file1 на отдельные файлы по 10 строк каждый, с именами Faa, Fab, ...;
   в) сортирует файл file1 в обратном порядке и помещает результат в файл file2.

Управление выполнением программ

1. Выводит на терминал «эхо-отображение» заданного текста (аргументы команды echo). Обычно применяется при слежении за выполнением командных файлов.

2. Сначала дать команду «kill —9 102», а затем «ps».

3. С помощью nice с аргументом в пределах от 1 до 20, где 20 означает установку низшего приоритета.

Информационные команды

1. С помощью команды «pwd».

2. Посредством команды «df /dev/usr».

3. Выполнить команду «date».

4. Дать команду «who am I».

5. ps a.

6. cal 1981. Если вместо 1981 вы дадите 81, то получите календарь 81 г. 1 века н. э.

Управление терминалом

1. stty 1200>/dev/tty3.

Обратите внимание на направление вывода, необходимое из-за того, что вы даете команду не для своего терминала.

2. Большинство печатающих устройств быстрее выводит символы табуляции, нежели пробелы.

3. tabs 1620.

К главе 8

1. Командный файл — то же, что и обычный, но содержит выполняемые команды UNIX или ed. Командный файл не требует опыта программирования, более того, он может применяться для решения задач, обычно требующих такого опыта.

2. Допускается 9 аргументов: $1, $2, ..., $9.

3. В качестве примера вложенных командных файлов можно предложить случай, когда файл A вызывает для выполнения файл B.

4. $MAIL, $HOME, $PATH.

К главе 9

1. Есть два привилегированных пользователя: root и bin. Пользователь root имеет доступ ко всем файлам, правила защиты информации для него недействительны. Пользователь bin управляет каталогами /bin и /usr/bin.

2. а) вызовите для редактирования файл паролей /etc/passwd;

б) добавьте информацию о новом пользователе в конец файла паролей;

в) создайте каталог для нового пользователя;

г) присвойте данный каталог новому владельцу;

д) установите права доступа каталога.

3. С помощью команды wall.

4. Дайте команду «ps ax!».

5. При отключении питания.

6. С помощью команды «df /dev/...» или «icheck /dev/...».

7. Поместите их в файл /etc/rc.

8. Путем изменения файла /etc/ttys. Если первый символ стро-

кн даиного термннала 0 — он отключеи, 1 — подключеи. Нужно перевестн систему в однопользовательскнй режнм, нзмеиить файл /etc/ttys н снова перейтн в многопользовательскнй режнм.

9. Индекс файла — это иомер нидексного дескрнптора; его можно рассматрнвать как внутреннее системиое имя файла нлн его идентифнкатор.

Обслужнванне н защита файловой системы

1. а) с помощью команды icheck файловая система проверяется на непротнворечивость спнска свободиых н спнска заиятых блоков;

б) команда ncheck выдает нмена файлов по задаиным номерам индексиых дескрипторов;

в) главная цель команды clri — удалеине файла, не встречающегося нн в одиом каталоге, а также удалеиие соответствующего ннднксного дескрнптора.

2. Файлы, содержащиеся в файловой системе (отлнчной от главной файловой системы), не могут нспользоваться до тех пор, пока она не будет монтирована.

3. Следует стать привилегнрованным пользователем, выполннв команду su.

4. Главная цель команды dump — созданне защнтиых копий файлов на случай выхода нз строя машины, в то время как команда tar служнт для сохранения файлов в нормальном режиме работы с последующнм их быстрым восстаиовленнем.

Приложение Г

ЛИТЕРАТУРА

1. Allshouse R. A., McClellan D. T., Prlne E. G. and Rolla C. P. CSDP as an ADA Environment. Ada Environment Workshop, DoD High Order Language Working Group, Nov. 1979, p. 113—125.
2. Baker F. T. Structured Programming in the Production Programming Environment. — Proc. Int'l Conf. Reliable Software, 1975, p. 172—185.
3. Barak A. B. and Shapir A. UNIX with Satellite Processors. Software-Practice & Experience. Vol. 10, No. 5, May 1980.
4. Bourne S. R. An Introduction to the UNIX Shell. — Bell System Technical J., vol. 57, No. 6, Oct. 1978, p. 2792—2822.
5. Brinch Hansen P. Structured multiprogramming. Prentice-Hall, Inc., Englewood Cliffs, N. J., , 1978.
6. Dolotta T. A., Haight R. C. and Mashey J. R. UNIX Time-Sharing System: The Programmer's Workbench. — Bell System Technical J., vol. 57, No. 6, Oct. 1978, p. 2177—2200.
7. Dolotta T. A. and Mashey J. R. An Introduction to the Programmer's Workbench. Proc. 2nd — Int'l Conf. Software Eng., Oct. 1976, p. 164—168.
8. Dolotta T. A. and Mashey J. R. Using a Command Language as the Primary Programming Tool. — In: Command Language Directions. Proc. 79 IFIP Working Conf. Command Languages, D. Beech, ed. North-Holland, Amsterdam, 1980.
9. Enslow P. H., Jr. Portability of Large Cobol Programs: The Cobol Programmer's Workbench. Georgia Institute of Technology, Atlanta, Ga., Sept. 1979.
10. Fieldman S. I. MAKE-A Program for Maintaining Computer Programs. UNIX Programmer's Manual, vol. 9, Apr. 1979, p. 255—265.
11. Glasser A. L. The Evolution of a Source Code Control System. SICSOFT, vol. 3, No. 5, Nov. 1978, p. 121—125.
12. Hall D. E., Scherrer D. K. and Sventek J. S. A Virtual Operating System. Comm. ACM, vol. 23, No. 9, Sept. 1980, p. 495—502.
13. Harland D. M. High Speed Data Acquisition: Runnig a Realtime Process and a Time-Shared System (UNIX) Concurrently. Software-Practice & Experience, vol. 10, No. 4, April 1980.
14. Ivie E. L. The Programmer's Workbench-A Machine for Software Development. Comm. ACM, vol. 20, No. 10, Oct. 1977, p. 746—753.
15. Jackson M. A. Principles of Program Design. Academic Press. London, 1975.
16. Johnson S. C. and Ritchie D. M. UNIX Time-Sharing System: Portability of C Programs and the UNIX System. — Bell System Technical J., vol. 57, No. 6, Oct. 1978, p. 2021—2048.
17. Joy W. N., Graham S. L. and Haley C. B. UNIX Pascal User's Manual. Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1977.
18. Kay A. and Goldberg A. Personal Dynamic Media. Computer, Mar. 1977, p. 31—41.

19. Kernighan B. W. and Mashey J. R. The UNIX Programming Environment. Software-Practice & Experience, vol. 9, No. 1, January 1979.
20. Kernighan B. W. and Plauger P. J. Software Tools. Addison-Wesley, Reading, Mass., 1976.
21. Kernighan B. W. and Ritchie D. M. The C Programming Language. Prentice-Hall, Englewood Cliffs, N. J., 1978. Русский перевод: Керниган Б., Ритчи Д., Фьюэр А. Язык программирования Си. Задачи по языку Си. — М.: Финансы и статистика, 1985.
22. Larmouth J. Scheduling for a share of the machine. — Software-Practice and Experience, 5, 29—49 (1974).
23. Lions J. An operation system case study. — Operating Systems Review, 12, No. 3, 46—53 (1978).
24. Lions J. Experiences with the UNIX Time-Sharing System. Software-Practice & Experience, vol. 9, No. 9, September 1979.
25. Lions J. The UNIX Operating System. Commentary, Bell Telephone Laboratories, Murray Hill, N. J., 1977.
26. Lycklama H. and Chistensen C. A minicomputer satellite processor system. — The Bell System Tech. Journal, 57, 6, part 2, 2102—2113 (1978).
27. Mashey J. R. Using a Command Language as a High-Level Programming Language. Proc. 2nd — Int'l Conf. Software Eng., Oct. 1976, p. 169—176.
28. McCauley E. J., Barksdale G. L. and Holden J. Software Development Using a Development Support Machine. ADA Environment Workshop, DoD High Order Language Working Group, Nov. 1979, p. 1—9.
29. McCauley E. J. and Drongowski P. J. KSOS — The Design of a Secure Operating System. — AFIPS Conf. Proc., 1979 NCC, June 1979, p. 345—353.
30. Miller Richard. UNIX-A Portable Operating System? — Operating Systems Rev., vol. 12, No. 3, July 1978, p. 32—37.
31. Pearson D. J. The Use and Abuse of a Software Engineering System. AFIPS Conf. Proc., 1979 NCC, p. 1029—1035.
32. Popek G. J. et al. UCLA Secure UNIX. AFIPS Conf. Proc., 1979 NCC, June 1979, p. 355—364.
33. Risenberg M. Software Costs Can Be Tamed, — Developers Told. Computerworld, Jan. 29, 1980, p. 1—8.
34. Ritchie D. M. The Evolution of the UNIX Time-Sharing System. Proc. Symp. Language Design and Programming Methodology. Sidney, 1979.
35. Ritchie D. M. UNIX Time-Sharing System: A Retrospective. — Bell System Technical J., vol. 57, No. 6, Oct. 1978, p. 1947—1969.
36. Ritchie D. M., Johnson S. C., Lesk M. E. and Kernighan B. W. UNIX Time-Sharing System: The C Programming Language. — Bell System Technical J., vol. 57, No. 6, Oct. 1978, p. 1991—2019.
37. Ritchie D. M. and Thompson K. The UNIX Time-Sharing System. — Comm. ACM, vol. 17, No. 7, July 1974, p. 365—375.
38. Robinson R. A. and Krzysiak E. A. An Integrated Support Software Network Using NSW Technology. AFIPS Conf. Proc., 1980 NCC, May 1980, p. 671—676.
39. Rochkind M. J. The Source Code Control System. IEEE Trans. Software Eng., vol. SE-1, No. 4, Dec. 1975, p. 364—370.
40. Snow C. R. The Software Tools Project. Software-Practice & Experience, vol. 8, No. 5, Sept. — Oct. 1978.
41. Stockenberg J. E. and Taffs D. Software Test Bed Support Under PWB/UNIX. ADA Environment Workshop, DoD High Order Language Working Group, Nov. 1979, p. 10—26.
42. Teichroew D. and Hershey III E. A. PLS/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems. — IEEE Trans. Software Eng., vol. SE-3, No. 1, Jan. 1977, p. 42—48.
43. Teitelman W. A Display Oriented Programmer's Assistant. CSL 77-3, Xerox Corp. Palo Alto Research Center, Palo Alto, Calif., Mar. 1977.

44. Teitelman W. INTERLISP Reference Manual, Xerox Corp. Palo Alto Research Center, Palo Alto, Calif., Dec. 1978.
45. Thompson K. The UNIX Command Language. In: Structured Programming — Infotech State of the Art Report, Infotech International Ltd., Berkshire, Mar., 1975, p. 375—384.
46. Thompson K. and Ritchie D. M. UNIX Programmer's Manual. 6th edn., Bell Telephone Lab., Murray Hill, N. J., 1975.
47. Wegner P. The ADA Language and Environment. Proc. Electro/80, Western Periodicals Co., North Hollywood, Calif., May 1980.
48. Woodward J. P. L. Applications for Multilevel Secure Operating Systems. AFIPS Conf. Proc., 1979 NCC, June 1979, p. 319—328.
49. Yourdon E. and Constantine L. L. Structured Design. Yourdon Press, London, 1975.

Приложение Д

Дополнение:

ПРИНЦИПИАЛЬНЫЕ ОСНОВЫ И ПЕРЕНОС UNIX

Операционная система UNIX заметно выделяется из ряда операционных систем последних лет, несмотря на то, что многие технические решения, примененные в ней, по отдельности встречались и в других системах. В обширной литературе, посвященной UNIX, она сопоставляется с наиболее распространенными системами, такими, как CP/M, MS/DOS, рассматриваются последние разработки UNIX-подобных систем, новые реализации языка Си и прогнозируется будущее UNIX (см., например, [1]).

Такое внимание к UNIX объясняется тем, что эта интересная разработка, без сомнения, составит определенный этап в развитии системного программирования.

И пользователю, и системному программисту UNIX представляется в виде очень простой системы. Простота была, по-видимому, одним из важнейших критериев при выборе того или иного решения. Основные свойства UNIX определяются тремя главными составляющими, на которых строится все здание системы:

языком реализации Си, обусловливающим мобильность системы;

файловой системой, унифицирующей все средства передачи информации в UNIX;

командным языком, поднимающим интерфейс пользователя с системой до уровня современных языков программирования.

Перенос UNIX на новые ЭВМ и разработка UNIX-подобных систем ведутся в различных странах, в том числе и в Советском Союзе [1], сфера применения системы непрерывно расширяется. Так, например, UNIX объявлена основной системой в английском проекте ЭВМ пятого поколения ALVEY [3].

Здесь мы рассмотрим принципиальные основы построения системы, т. е. вопросы, практически не нашедшие отражения в книге Р. Готье. Кроме того, мы уделим внимание проблемам переноса системы, проблемам, связанным с разрядностью ЭВМ и размером виртуального адресного пространства, представлением символьной информации и переходом на национальные алфавиты, модификации системы при ее переносе.

1. ПРОЦЕССЫ

UNIX — мультипрограммная система. Работы, выполняемые в системе, представлены множеством конкурирующих процессов. Процесс строго последователен, и на невозможности выполнять внутри процесса асинхронные действия построены многие механизмы управления в системе. Даже ввод-вывод не может быть выполнен процессом асинхронно: инициируя физическую операцию ввода-вывода, процесс ждет ее завершения и лишь после этого сможет продолжить работу.

Процесс — единица управления и потребления ресурсов в системе. Распределение ресурсов между процессами, в частности предоставление им центрального процессора, выполняет ядро системы. Процессы работают в режиме «пользователь», ядро — в привилегированном режиме «система» (на PDP-11 это режимы «user» и «kernel» соответственно).

Ядро системы (кроме программ реакций на прерывания) не самостоятельная вычислительная единица. Программы ядра выполняются от имени процессов, их вызвавших. Процесс может находиться в одной из двух фаз. Работая в режиме «пользователь», т. е. выполняя пользовательскую программу, процесс находится в пользовательской фазе. При выполнении некоторой программы ядра он делает так называемый системный вызов (запрос к ядру). С точки зрения синтаксиса на языке Си это выглядит как вызов функции, реально же происходит прерывание, и процесс начинает выполнять требуемую программу ядра в режиме «система». Работая в этом режиме, процесс находится в системной фазе.

Например, давая системный вызов read (чтение файла), процесс входит в соответствующую программу ядра. Последовательно вызывая различные функции ядра, обрабатывающие этот системный вызов, процесс входит, наконец, в драйвер требуемого устройства. Таким образом, драйвер тоже выполняется от имени процесса, затребовавшего обращение к устройству. Синхронность ввода-вывода и выполнение ядра от имени процессов оказываются взаимосвязанными свойствами системы.

Некоторые участки системных фаз процессов являются критическими в том смысле, что пока процесс не выйдет из участка, другой процесс не должен в него входить. Критические участки необходимы для обеспечения целостности данных ядра: если некоторый процесс начал модификацию какой-либо структуры (например, индексного дескриптора файла), то пока модификация не закончится, к работе с этой структурой не может быть допущен другой процесс.

Синхронизация системных фаз процессов на границах критических участков реализуется в ядре аппаратом событий. В UNIX этот аппарат предельно упрощен. Процессы, ждущие наступления событий, снимаются с ожидания процессом, который это событие объявляет. Сам факт наступления события не фиксируется

ни в каком бите и после актнвнзацнн процессов, ждавших этого события, от последнего не остается никакого следа.

Такая схема взаимодействия процессов соответствует механизму сопрограмм. Использование сопрограмм упрощает логику ядра системы и требует одного выделенного процесса, создающегося нестандартным образом. С него начннается работа снстемы после запуска. В UNIX этот процесс называется днспетчерским и не имеет пользовательской фазы.

Все процессы в UNIX (кроме днспетчерского) создаются с помощью операцнн порождення. В порожденин участвует пара процессов: порождающий («отец») и порожденный («сын»). Порождающий процесс выполняет системный вызов fork; в результате появляется порожденный процесс.

Каждый процесс имеет уникальный целочисленный идентификатор (или номер). Идентификатор присваивается процессу при его порождении и служит затем единственным понятием, однозначно идентифицирующим процесс. В командах или системных вызовах, где нужно указать процесс, задается идентификатор требуемого процесса. Идентификацня процесса никак не связана с именем программы, выполняемой процессом, благодаря чему процесс может последовательно выполнять различные программы.

При начальном запуске системы первым, как уже говорилось, начинает работать диспетчерский процесс. Его создает ядро системы и присваивает ему ндентификатор 0. В дальнейшем в функции этого процесса будет входить реализация свопинга, т. е. загрузка образов процессов в оперативную память и выгрузка образов тех процессов, которые (на основе определенной стратегии использования памяти) должны эту память освободить.

Диспетчерский процесс порождает процесс с идентификатором 1, выполняющим специальную программу init. Она анализирует содержимое файла терминальной конфигурацин /etc/ttys. Для каждого терминала, логически подключенного согласно этому файлу к системе, init порождает процесс. Этот процесс запрашивает с терминала имя пользователя, а еслн нужно, то и пароль, и, проверив корректность полученной информации, вызывает интерпретатор команд shell. Таким образом, после начального запуска системы появляются процессы с идентификаторами 0, 1 и по одному процессу для каждого терминала. С помощью интерпретатора shell пользователи могут теперь начать диалог с системой.

UNIX — система разделения времени. Хотя процессам приписывается прноритет, он не нграет здесь той решающей роли, что в системах реального временн, например RSX-11. Доступ к ресурсам осуществляется не на приорнтетной основе; в частности, не упорядочены по приоритету процессы, претендующие на процессор. Алгорнтм диспетчеризации устроен так, что процессор достается процессу, имеющему наивысший приоритет. На практике же все процессы, кроме диспетчерского, имеют одинаковый при-

оритет, так что действует правило циклического квантования времени.

От системы реального времени UNIX отличает также отсутствие развитых средств взаимодействия процессов. Один процесс может послать другому сигнал — так реализуется логическое взаимодействие процессов в UNIX. Но чтобы послать сигнал, нужно знать идентификатор принимающего процесса, а для этого быть его «родственником» (см. об этом ниже). Информационное взаимодействие процессов — механизм программных каналов — также основано на родственных связях. Какого-либо общего аппарата взаимодействия и синхронизации процессов в UNIX нет.

2. УПРАВЛЕНИЕ ПАМЯТЬЮ

Организация виртуальной памяти, как, впрочем, и многое другое, решена в UNIX очень просто. С определенной точки зрения даже слишком просто, и в некоторых UNIX-подобных системах наблюдается возврат к традиционным, более громоздким решениям.

Каждый процесс в UNIX работает в своем собственном виртуальном адресном пространстве. Образ процесса (совокупность участков памяти, отображаемых виртуальными адресами) состоит из трех сегментов и стека. Стек используется для организации работы с подпрограммами и других вычислений, где удобен принцип действия стека. Сегменты соответствуют программе, выполняемой процессом. Допустимы три сегмента: процедурный, динамический и сегмент данных.

Процедурный сегмент содержит машинные инструкции и константы. Сегмент данных и динамический сегмент содержат данные, причем в первый входят данные, инициируемые при компиляции, а во второй — данные, неинициируемые при компиляции. В этом смысле сегмент данных — статический. Выполняемый файл — результат работы редактора связей ld — содержит только процедурный сегмент и сегмент данных. Место под динамический сегмент и стек выделяется при загрузке образа процесса в оперативную память.

Машинные инструкции выделены в отдельный сегмент для того, чтобы можно было работать с реентерабельными программами. Если программа специальным флагом команды Ed объявлена реентерабельной, процедурный сегмент разделяется всеми процессами, выполняющими эту программу. Причем в оперативной памяти находится один экземпляр процедурного сегмента, который входит в виртуальные пространства всех этих процессов.

Таким образом, в структуре программы нет секций — традиционного понятия многих операционных систем и языков Ассемблера. Обычно секция служит единицей компоновки, а сегмент — единицей загрузки. В UNIX это не так. Единицей компоновки является сегмент (процедурный, сегмент данных или динамический),

а единицы загрузки, прнчем загрузки в виртуальную память, нет вообще. Счнтается, что образ всего процесса должен помещаться в имеющееся виртуальное адресное пространство.

По большей частн решення, принятые в UNIX, направлены на уменьшение размера используемых программ (о чем в условиях большого внртуального пространства можно даже не заботиться). Однако на PDP-11, где длина виртуального адреса 16 разрядов, а размер виртуального пространства — 64 кнлобайта, ограничение размера программы размером виртуального пространства становится существенным. Не для всех программ 7-й версии UNIX это требование выполняется, так что 7-я версия в полную силу работает на тех моделях PDP-11, где есть два адресных пространства: пространство инструкций и пространство данных.

Обычно, когда не хватает диапазона виртуальных адресов, используется оверлейная структура программы, как в нзвестных операционных системах фирмы DEC для PDP-11 — RSX-11 и RSTS(E). Но в UNIX оверлейных структур нет. Во-первых, без оверлейных структур намного проще как пользователям, так и системе, а простота — однн из важнейших критернев при выборе тех или иных решений в UNIX. Если оверлейные структуры допускаются, пользователь должен описать такую структуру спецнальными языковыми средствами, а система должна понимать это описание и строить программу соответствующим образом. В RSX-11 есть специальный язык описания перекрытий ODL, на котором пользователь опнсывает оверлейную структуру своей программы. Интерпретирует этот язык и строит программу (образ задачи) построитель задач TKB. Программа TKB очень громоздка, по сравнению с ней редактор связей ld в UNIX нсчезающе мал. Во-вторых, наличие оверлейных структур противоречнт требованию мобильности системы. Оверлейный механизм, как сказано выше, не прозрачен для пользователя: он явно проявляется в интерфейсе пользователя с системой, например, в виде языка описания перекрытий. Но в мобильной системе этот интерфейс должен быть максимально машинно-независнмым, иначе мобильность не нмеет смысла. Оверлейный механизм, призванный экономить внртуальную память, нужен там, где этой памяти не хватает (в 16-разрядных ЭВМ, например, PDP-11) и не нужен в противном случае (в 32-разрядных ЭВМ, например, VAX-11).

Значит, проблемы, возникающие из-за малостн днапазона виртуальных адресов, машинно-зависимы. Любое решение таких проблем, влияющее на интерфейс пользователя с системой, также машинно-зависимо. Оверлейный механизм попадает как раз в эту категорню.

Отсутствие в UNIX оверлейных структур и есть то слишком простое решение вопросов, связанных с виртуальной памятью, о которых шла речь в начале этого раздела. Теоретнчески все логично. Практически втиснуть все программы в узкие рамки 16-разрядной адресации затруднительно.

Требование разместнть программы в пределах внртуального

пространства в UNIX не простая декларация. Оно подкрепляется рядом технических решений. Если программа большая, ее можно разбить на части (фазы) и выполнять их либо последовательно в рамках одного процесса, либо асинхронно — как разные процессы. Однако в основном уменьшение размеров программ достигается в UNIX другим путем.

Как правило, объем программы сильно возрастает не за счет собственно кода (машинных инструкций) или скалярных данных, а за счет больших массивов данных. Имеено большие массивы приводят к тому, что суммарно программа и данные не умещаются в виртуальном пространстве. Если пространства разделены — одно для инструкций, другое для данных — ситуация улучшается, но при достаточно больших объемах массивов данные могут не поместиться в отведенное им адресное пространство.

Кардинальное решение проблемы — в исключении массивов из виртуального пространства. В UNIX это делается следующим образом: большие массивы должны рассматриваться как файлы. Иными словами, к массиву следует обращаться не с помощью обычных машинных инструкций, а с помощью операций, предназначенных для работы с файлами.

На первый взгляд такое решение не эффективно. Действительно, если массив находится в виртуальном пространстве, к его элементу можно обратиться с помощью обычных машинных инструкций, потратив считанные микросекунды или десятки микросекунд. Обращение к файлу предполагает прерывание, обработку системного вызова в ядре, обращение к диску и лишь затем пересылку требуемой информации из памяти процесса в системный буфер. Ясно, что при втором варианте времени на обращение уйдет больше.

В UNIX приняты меры к устранению недостатков, возникающих при использовании массивов в качестве файлов. Оптимизация касается как доступа к файлу в целом, так и доступа к его элементам. В системе применена тотальная буферизация ввода-вывода, основанная на принципах, близких принципам построения кэш-памяти. Системные буфера объединяются в программно реализованную кэш-память, и весь обмен между внешними устройствами и памятью процессов буферизуется в этой кэш-памяти. Всякий раз, когда нужен какой-либо блок диска или магнитной ленты, проверяется наличие его в кэш: если блок уже находится в кэш, обращение к внешнему устройству не происходит. Это оптимизирует доступ к блокам файлов.

Оптимизация обращения к элементу файла основана на максимальном упрощении его структуры. С точки зрения ядра система файл — это одномерный массив байтов с прямым доступом. Никакой структуры записей внутри файла не предполагается. Обмен с файлом происходит последовательностями байтов, что требует минимальных системных издержек на чтение из файла или запись в него.

Представление массивов в виде файлов и оптимизация досту-

па к файлам — это вопросы, которые необходимо решать ввиду отсутствия оверлейных структур в системе. Однако, несмотря на то, что оверлейный механизм сомнителен с принципиальной точки зрения, он привлекателен с точки зрения практического применения. Видимо, поэтому оверлейные структуры введены в некоторых UNIX-подобных системах, например в операционной системе V7M-11 — варианте UNIX фирмы DEC для всех моделей PDP-11 и Micro/PDP-11 с диспетчером памяти.

3. ФАЙЛОВАЯ СИСТЕМА

Файловая система занимает чуть ли не центральное место в UNIX и определяет многие привлекательные стороны системы. Существенно, что файловая система не является надстройкой над какими-то базовыми понятиями операционной системы, как это зачастую бывает, а, наоборот, понятия, связанные с файловой системой, в UNIX первичны и создают базис для других, более высоких системных понятий. Файловая структура известна ядру системы, механизм работы с файлами (файловый механизм) встроен в ядро и составляет значительную его часть.

Файловая система UNIX заимствована из операционной системы MULTICS [4], с которой работали создатели UNIX К. Томпсон и Д. Ритчи. От MULTICS идет иерархия каталогов, понятие текущего каталога, монтирование файловых систем. В UNIX все это приобрело новые свойства и оказалось удачно увязано с другими системными конструкциями и понятиями.

3.1. СТРУКТУРА ФАЙЛОВОЙ СИСТЕМЫ

Файловая система UNIX имеет иерархическую структуру, обычно называемую деревом. В узлах дерева расположены каталоги; содержимое каталога — это перечень других файлов и каталогов, к которым можно провести дугу из данного каталога. Корнем дерева служит каталог, в который не входит ни одна дуга, а листья — файлы, не каталоги.

Каталоги тоже являются файлами. Их именуют и работают с ними как с любыми другими файлами, за одним исключением: запись в каталог может делать только ядро системы. Это естественно. Файловая система — базовое понятие для UNIX, файловую структуру понимает ядро, и ядро, следовательно, отвечает за целостность файловой структуры, вот почему запись в каталог выполняется под строгим контролем ядра.

В такой структуре естественно связать путь в графе с именем файла. Полным именем файла называется последовательность имен каталогов, стоящих на пути от корня дерева к этому файлу. Начинается последовательность с наклонной черты /, заканчивается именем данного файла в последнем каталоге; имена разделяются наклонной чертой. Например, имя /usr/bin/zet означает,

что usr — подкаталог корневого каталога, bin — подкаталог каталога /usr, а zet — файл в каталоге /usr/bin.

Файловую структуру UNIX действнтельно можно было бы считать деревом, если бы от корня к каждому файлу вел единственный путь. Инымн словамн, тогда каждый файл имел бы единственное имя. Но это не так. К файлу может вести несколько путей. Например, упомянутый выше файл /usr/bin/zet может иметь второе нмя /al, т. е. согласно второму нменн он указан непосредственно в корневом каталоге.

Многократное нменование одного н того же файла принципиально возможно, но встречается редко, поэтому с учетом приведенной оговоркн файловая структура в UNIX может называться деревом. Примером использования второго имени (снноннма) для файла будет работа средств системной печати (спулинга). Пользователь отправляет файл file на системную печать командой

```
lpr file
```

Программа lpr создает синоним для файла file в каталоге /usr/lpd, т. е. новое имя /usr/lpd/file, для того, чтобы следящий процесс lpd, выполняющий системную печать, мог просматривать некоторый выделенный каталог в поисках файлов, подлежащих системной печатн. Таким каталогом и служит /usr/lpd.

Одни только полные имена файлов неудобны по двум причинам. Во-первых, если путь достаточно велик, писать полное имя долго. Во-вторых, указание полного имени заставляет ядро просматривать все каталогн, находящиеся на пути к файлу, что вызывает лишние обращения к дискам и в конечном счете снижает эффективность системы. Хорошо было бы пользоваться относительными именами, указывая путь к файлу не от корня, а от некоторого каталога, более близкого к файлу.

Так возникает понятие текущего каталога. Если каталог /usr/bin объявлен текущим, то файл /usr/bin/zet можно указывать просто по имени zet. Отсутствие наклонной черты в начале имени файла означает, что имя указано относительно текущего каталога.

Каждый диск имеет свою файловую структуру. Чрезвычайно важная операция монтирования позволяет объединять деревья в общее дерево файлов. Первоначально после запуска системы имеется (т. е. известно ядру UNIX) только дерево файлов на системном диске. Это — так называемая основная (немонтированная) файловая снстема. Затем монтируются другие диски (файловые системы), подключаясь к существующему общему дереву файлов в качестве поддеревьев. Монтирование — это выделение некоторого файла в существующем дереве и объявление его корневым каталогом подсоединяемого дерева.

Понятия текущего каталога и монтированной файловой системы тесно связаны между собой. Они позволяют при работе с

файлами абстрагироваться от полных имен файлов. У пользователя есть свой диск, содержащий его рабочие файлы. Администратор системы моитирует диск пользователя, последиий входит в систему, получает в качестве текущего иекоторый каталог на своем диске и имеиует свои файлы отиосительно этого каталога.

3.2. ТИПЫ ФАЙЛОВ. ВВОД-ВЫВОД

С точки зреиия ввода-вывода каталоги будут такими же файлами, как и обычиые файлы данных. Однако листьями дерева могут выступать ие только обычиые файлы, но и так иазываемые специальиые файлы. Этим термином обозначены в UNIX внешние устройства. Специальные файлы защищаются и обрабатываются так же, как обычиые файлы и каталоги. В UNIX различаются эти три иазванные типа файлов.

Обычиые файлы являются обязательно дисковыми файлами. Файловая структура иа магиитной леите не поддерживается системой. Леита целиком может рассматриваться как файл, т. е. как специальный файл. Идея специального файла и состоит в том, чтобы работать с внешним устройством целиком как с файлом, т. е. с дисковыми (обычными) файлами и в то же время со всем диском тоже как с файлом (специальным файлом).

Сама по себе унификация назваиий иичего ие дает. Если общность в иазваиии ие будет подкреплена общностью в сопутствующих поиятиях и средствах, ии пользователю, ии системе легче не станет. В UNIX такая общиость подкрепляется соответствующей организацией механизмов защиты файлов и ввода-вывода.

Как уже было сказаио, все файлы, независимо от типа, защищаются одииаково. В книге Р. Готье код защиты файлов рассматривался достаточио подробно. Следует остановиться лишь иа процедуре записи в каталог. Явиой операцией записи пользователь не может изменить информацию в каталоге: реальиую запись, как уже говорилось, может выполнять только ядро. Такая запись требуется ядру при создаиии или удалеиии файлов, и разрешеиие пользователю делать запись в каталог озиачает, что он может создавать и удалять файлы в этом каталоге.

Ввод-вывод в UNIX реализуется иабором системиых вызовов. Файл открывается системиым вызовом open, создается заново — системным вызовом creat. Для чтения и записи имеются системные вызовы read и write. Указатель чтения/записи может быть установлеи в произвольиую позицию системным вызовом lseek, что символизирует прямой доступ к файлу.

Уиификация ввода-вывода по отношению к различным типам файлов дает совершенио одинаковые записи для этих системных вызовов при любом типе файла. Например, файл с именем name открывается системиым вызовом (иа языке Си).

```
fd=open (name, mode)
```

где аргумент mode указывает, для чтения, записи или модификации открывается файл. Имя файла name может быть записано в любой требуемой форме. Так, вызов

```
fd=open ("/usr/bin/ex", 1)
```

открывает для записи обычный файл /usr/bin/ex,
вызов

```
fd=open ("/usr/bin/", 0)
```

открывает для чтения каталог/usr/bin, а вызов

```
fd=open ("/dev/rko", 2)
```

открывает для чтения и записи диск rk3 как один (специальный) файл.

Системный вызов open создает в ядре для открытого файла структуру данных, называемую дескриптором файла. Процесс может открыть не более 20 файлов. Системный вызов open возвращает номер дескриптора файла, который присваивается переменной fd.

Номер дескриптора используется затем в операциях чтения и записи. С точки зрения ядра системы файл — одномерный массив байтов с прямым доступом. Поэтому вызовы read и write передают последовательность байтов. Вызов

```
read (fd, A, n)
```

читает из файла n байтов и записывает в память процесса по адресу A. Чтение начинается от текущего указателя чтения/записи в файле, после чего указатель смещается на n байтов вперед по файлу. Файл должен быть открыт и задается номером дескриптора fd.

Системный вызов write имеет те же аргументы, только осуществляет запись в файл.

Все эти операции выполняются с открытым файлом и, следовательно, применимы к любому типу файла. Имя и тип файла указываются только при открытии, в дальнейшем используется дескриптор файла. Естественно, что процесс, открывающий файл, должен иметь соответствующие права доступа к нему.

Представление файла в виде массива байтов с прямым доступом задает простейшую структуру файла. Отсутствие внутризаписной структуры и ввод-вывод в терминах последовательностей байтов минимизируют время, затрачиваемое на чтение и запись. С другой стороны, использование больших массивов в качестве файлов приводит к экономии виртуальной памяти в UNIX, и при простой структуре файлов становится эффективнее. Таким образом, представление файлов в виде массивов байтов и представление массивов в виде файлов тесно связаны одно с другим и взаимнообусловлены.

3.3. СПЕЦИАЛЬНЫЕ ФАЙЛЫ

Специальные файлы позволяют с помощью обычных файловых операций обрабатывать внешние устройства. В UNIX допускается две разновидности специальных файлов: блокориентированные и байториентированные. Блокориентированный интерфейс дает возможность обмениваться с магнитными дисками и лентами как с устройствами, состоящими из блоков. Байторнентнрованный интерфейс позволяет обмениваться со всеми внешними устройствами, рассматривая их как массивы байтов.

Блокориентированный интерфейс определяет некоторые принципиальные особенности системы. Во-первых, физический обмен с устройством выполняется поблочно, т. е. сравнительно небольшими порциями за один раз. Во-вторых, информация, подлежащая вводу-выводу, подвергается тотальной системной буферизации. В ядре системы выделяется уже упоминавшаяся программно организованная кэш-память. При вводе блок читается с устройства в кэш, а затем в память процесса передается число байтов, запрошенное процессом в системном вызове read. Прн выводе информация пересылается из памяти процесса в кэш, а оттуда уже на устройство.

Пул буферов, выделенный в ядре для буферизации блокориентированного ввода-вывода, не случайно назван кэш-памятью. Хотя эта память организована программно и не имеет прямого доступа (буфера просматриваются последовательно при поиске нужного блока), по отношению к диску она играет ту же роль, что и аппаратно реализованная кэш-память по отношению к оперативной памяти. Когда требуется прочесть блок с диска, сначала просматриваются кэш на предмет наличия этого блока в ней. Если блок найден, физического обращения к диску не происходит. Иначе блок читается с диска в кэш и при последующих обращениях к нему будет уже браться из кэш.

С кэш связан один весьма существенный недостаток системы. Когда кэш строится аппаратно и ставится, например, между процессом и оперативной памятью, запись в память всегда реализуется в виде записи как в кэш, так и в оперативную память. Поэтому в любой момент времени содержимое кэш идентично содержимому соответствующих ячеек оперативной памяти. Если кэш выйдет из строя, целостность информации в оперативной памяти не нарушится.

В UNIX — иначе. При записи в файл выводимые байты попадают в кэш и не пересылаются на диск, даже если буфер в кэш полностью сформирован и готов к выводу. Причины понятны: система и так не блещет хорошей реактивностью и лишние обращения к дискам только ухудшат эту достаточно важную ее характеристику. Поэтому информация из кэш выводится на диски только в двух случаях: потребовался какой-либо блок и все буфера в кэш заняты или по системному вызову sync, специально предназначенному для вывода задержавшейся в кэш информации.

Таким образом, блокориентированный нитерфейс предполагает сравнительно медленный способ обмена (поблочно), но минимизирует число обменов за счет анализа кэш-памяти. В противоположность блокориентированному байториентированный нитерфейс дает более быстрый способ обмена, но не нуждается в кэш. Байториентированный интерфейс для магнитных дисков и лент называется в UNIX прозрачным вводом-выводом. Прозрачный обмен осуществляется непосредственно между памятью процесса и внешним устройством. Порции передаваемой информации в этом случае могут быть существенно больше одного блока.

В некоторых специальных случаях прозрачный интерфейс очень удобен. Но вообще говоря, этот способ обмена менее корректен, чем блокориентированный. Если процесс непосредственно взаимодействует с внешним устройством, он не может быть выгружен из оперативной памяти при свопинге. Системная буферизация, построенная на принципах кэш-памяти, не препятствует свопингу процессов и делает блоки, присутствующие в кэш, доступными всем процессам.

Основные преимущества прозрачного интерфейса перед блокориентированным — физический обмен большими порциями информации. Чтобы умалить это преимущество и повысить эффективность блокориентированного обмена как более корректного интерфейса, в последней версии системы — UNIX Sysetm5 — величина блока, т. е. единицы обмена, доведена до 1 килобайта. Если объем оперативной памяти не менее мегабайта и диски достаточно быстрые (именно так обстоит дело на PDP-11/44 и 70, для которых предназначена эта версия), увеличение единицы обмена вполне целесообразно.

4. КОМАНДНЫЙ ЯЗЫК

Говоря о командном языке UNIX, Р. Готье описывает лишь самые элементарные возможности интерпретатора командного языка — программы shell. А между тем командный язык UNIX (иногда его называют shell по имени интерпретатора) — одна из изюминок системы. Без него система потеряла бы значительную часть своей привлекательности.

Командный язык UNIX появился не на пустом месте. В нем синтезированы достоинства языков управления заданиями в пакетных системах, командных языков и командных файлов интерактивных систем, а также современных языков программирования. Сочетание средств, необходимых для командного языка, и свойств языков высокого уровня обусловило практически универсальные возможности программирования и управления в рамках единого языка.

Внешне командный язык UNIX — это современный язык программирования, позволяющий использовать переменные, различные операторы управления, группировать операторы в более сложные конструкции. Программу, написанную на командном

языке, выполняет (в режиме интерпретации) интерпретатор команд shell.

Единица действия в командном языке — простая команда. Примеры простых команд и с достаточно подробным их описанием даны в книге. Формально простая команда определяется как последовательность слов, разделенных промежутками. Слово не содержит пробелов, табуляций и переводов строки. Это символы, из которых (по умолчанию) строятся промежутки между словами.

Первое слово команды служит ее именем. Интерпретатор shell не знает списка всех команд, т. е. не различает имена команд. Имя команды должно быть именем файла. Shell находит этот файл и организует его выполнение, что и означает выполнение команды.

Из приведенного правила есть одно исключение. Существуют так называемые встроенные команды, выполняющиеся самим интерпретатором. Имена этих команд shell знает. Встроенными будут, например, команды cd и login, описываемые в книге Р. Готье. Ниже мы объясним необходимость встроенных команд, а сейчас рассмотрим организацию выполнения файлов в случае, когда команда не встроенная и имя команды есть имя файла.

Для выполнения файла shell порождает процесс, который и будет выполнять файл. Прежде чем порожденный процесс вызовет файл для выполнения, shell анализирует тип файла: выполняемый он или текстовый.

Выполняемый файл — обычная программа. Текстовый файл — это командный файл. В первом случае порожденный процесс будет выполнять указанный файл. Во втором — порожденный процесс должен выполнять интерпретатор команд shell, интерпретирующий указанный командный файл. Анализируя команду, shell должен различать эти случаи и по-разному организовывать выполнение команды.

Для определения такого различия в некоторых файлах с помощью первого слова задается тип файла. В этом слове размещается так называемый системный код файла (magic number). В частности, системный код выполняемого файла равен одному из восьмеричных чисел: 407, 410, 411, 405. В принципе текстовый файл может начинаться с такого слова. Но в любом случае нужно, чтобы файл имел разрешение для выполнения, а обычный текстовый файл, по умолчанию, его не имеет. Следовательно, если файл, доступный для выполнения, обладает указанным выше системным кодом — это двоичная программа. Иначе — командный файл, обрабатываемый интерпретатором shell.

Shell анализирует системный код файла по той причине, что и двоичная программа, и командный файл синтаксически вызываются одинаково, в форме.

```
alpha a1 a2 ...
```

где alpha — имя файла; a1, a2 и т. д. — аргументы команды. Если alpha — командный файл, то предыдущая запись эквивалентна

```
sh alpha a1 a2 ...
```

Здесь явно указан вызов двоичной программы sh (интерпретатора shell), а alpha выступает в роли аргумента. И такая запись допустима в командном языке, но первая форма удобнее: она позволяет не различать вызов двоичной программы и вызов командного файла. Сравнительно небольшое синтаксическое удобство в сочетании с алгоритмическими возможностями командного языка позволяет легко расширять набор команд путем создания дополнительных командных файлов. Уже тот факт, что именем комаиды служит имя файла, существеино облегчает расширение набора команд — включение в файловую систему нового файла означает появление иовой комаиды. Но не обязательно писать иовую программу (иовый выполняемый файл) на языке Си, Фортран и т. п. Мы можем составить (если это удается) командный файл, разрешить его выполиять и также получим новую команду.

В командных файлах UNIX в отличие от других операционных систем (например, RSX-11) употребляются те же синтаксические конструкции, что и при вводе комаид с терминала, поэтому комаидные файлы часто называют процедурами shell. В ряде коикретных применений даже злоупотребляют этими возможностями командного языка, отдавая ему предпочтение перед обычными языками программирования. Нужно, однако, всегда иметь в виду: расширеиие набора команд за счет новых процедур shell — способ удобный, но вряд ли эффективный. Комаидный язык является интерпретируемым, а ие компилируемым языком, так что интерпретация командного файла — дело достаточно медленное. Использоваиие процедур shell в качестве команд системы должно оставаться приятной возможностью, реализовывать которую нужно в редких случаях, причем эффективность системы не должиа уменьшаться.

Возвратимся теперь к встроенным комаидам и рассмотрим вопрос их необходимости. По определению, если команда не встроенная, оиа выполияется процессом — сыиом процесса, выполияющего shell. Допустим, мы хотим изменить какой-либо атрибут иитерпретатора shell или выполняющего его процесса или как-либо повлиять на его работу. Тогда искомое действие должно выполияться самим интерпретатором, иначе, влияя на характеристики процесса, оно затронет его сына.

Например, команда wait позволяет приостановить диалог до завершения всех асиихронно запущеиных процессов. Реализуется она с помощью системного вызова wait, по которому процесс приостаиавливается до завершения своих сыновей. Ясно, что этот системный вызов должен выполнять сам shell, а не его сын, поскольку асинхронно запущенные процессы ие являются потомка-

ми этого сына. Команда wait, следовательно, должна быть встроенной.

Аналогичные замечания можно сделать и по поводу команды cd, меняющей назначение текущего каталога. У каждого процесса есть свой текущий каталог, который наследуется при порождении. Ясно, что новое назначение должно быть сделано для процесса, выполняющего shell, и от него оно будет унаследовано процессами, выполняющими команды.

Простые команды могут объединяться в более сложные конструкции. Кроме того, существуют операторы for, while, until, if, case с семантикой, соответствующей этим операторам в современных языках программирования. Для примера рассмотрим оператор for (он встретится нам дальше). Его общий вид:

```
for name in word ...
do list
done
```

name — переменная, служащая параметром цикла; word... — некоторая последовательность слов; list — список команд. Семантика оператора for очевидна: параметру цикла name последовательно присваиваются слова из конструкции word... и каждый раз выполняется список команд list. Выполнение цикла завершается после исчерпания слов, стоящих между in и do.

4.1. ПЕРЕМЕННЫЕ И КОМАНДНАЯ СРЕДА

Переменные (командные) определяются и получают значения в результате выполнения операторов присваивания. Существенно, что переменные в командном языке имеются, но типов переменных нет. Точнее, в языке есть один единственный тип — символьный, так что значениями переменных будут строки символов. Оператор

```
x1=a2t
```

присваивает переменной x1 строку из трех символов: a2t.

В языке нет операций над строками и переменными, например, нет явной операции конкатенации (соединения) строк символов. Однако неявно она возможна благодаря механизму подстановки (замещения). Существуют две разновидности подстановок: подстановка значения переменной и подстановка вывода команды. Рассмотрим первую из них. Простое использование в команде имени переменной не означает подстановку ее значения. Подстановка происходит, если имени переменной предшествует символ $. Можно сказать, что $ — унарная (причем единственная) операция, которая может быть применена к переменной. Она позволяет заменить строку символов — имя переменной, строкой символов — значением переменной.

Удобно проиллюстрировать это с помощью команды echo. Она просто выводит (в стандартный файл вывода) свои аргументы. Команда

```
echo .x1
```

выводнт строку «x1», а команда

```
echo ¤x1
```

выводит строку «a 2t».

Если переменным x1 и x2 присвоены какие-либо значення, то для конкатенацни этнх значений (конкатенацин строк) нужно напнсать

```
¤x1¤ x2
```

Командные переменные в UNIX — частный случай общего понятия «параметра». Прн вызове командного файла (процедура shell) аргументы, передаваемые в процедуру, также будут параметрами. Чтобы достичь снntаксического разлнчия между переменными и аргументамн вызова процедуры, введем следующее правило: имя переменной не может начинаться с цнфры, тогда как параметр, представляющий в теле процедуры аргумент ее вызова, обозначается с помощью цифры. Напрнмер, при вызове процедуры

```
prog x ab pi
```

аргументы x, ab н pi доступны в теле процедуры prog по нменам ¤1, ¤2 н ¤3 соответственно. Снмвол ¤ перед цнфрой необходим для подстановкн значення параметра.

Аппарат подстановкн значений параметров в командном языке UNIX достаточно велик. Есть спецнальные обозначения (нмена) для спецнальных параметров.

Например, значеннем параметра # служнт чнсло аргументов вызова процедуры. Часто встречается параметр *. Его значение — последовательность аргументов вызова, разделенных пробелами. Для вызова

```
prog x ab pi
```

в теле процедуры prog конструкция ¤# заменяется цифрой 3, а конструкция ¤* — последовательностью

```
«x ab pi»
```

Параметры типа ¤n (n — цифра в диапазоне от 1 до 9), представляющие в теле процедуры аргументы ее вызова, называются также позиционными параметрами. То, что n$\leqslant$9, означает, что одновременно доступны только первые девять. Доступ к остальным параметрам дает встроенная команда shift. Она переименовывает параметры так, что ¤2 становнтся ¤1, ¤3 — ¤2 и т. д. Эта команда удобна в циклах, так как позволяет переименовывать последовательно все параметры в ¤1. Когда спнсок аргументов вызова будет исчерпан, параметр ¤1 получит в качестве значення пустую строку. Ниже мы приведем еще однн пример употребления команды shift.

Конструкция in word... в общем виде оператора for может отсутствовать, вследствие чего переменная name как параметр цикла последовательно получает значения позиционных параметров, т. е. сначала $1, потом $2 и т. д. Иными словами, параметр цикла пробегает список аргументов вызова процедуры, так что их удобно обрабатывать в цикле.

Например, процедура mou, имеющая вид

```
for i
do /etc/mount /dev/ $i / $i
done
```

монтирует сразу несколько устройств (команда mount монтирует только одно устройство). В частности, команда

```
mou rk1 rp1 rp2
```

монтирует диски /dev/rk1,/dev/rp1 и и dev/rp2, делая файлы /rk1, /rp1 и /r2 корневыми каталогами монтированных файловых систем.

Команда shift, цикл без конструкции in word и другие средства обработки позиционных параметров были бы недоступны вне тел процедур, но встроенная команда set позволяет их использовать. Она дает возможность присвоить значения позиционным параметрам и пользоваться ими, как если бы они были переданы в вызове процедуры.

Например, команда

```
set x ab pi
```

присвоит свои аргументы параметрам $1, $2, $3 и сформирует значения параметров $#, $* так же, как при вызове

```
prog x ab pi
```

(см. выше).

Команда set удачно сочетается со второй разновидностью подстановок — подстановкой вывода команды, причем стандартный вывод команды можно направить, куда потребуется. Подстановка производится заключением команды в знаки слабого ударения. Конструкция

```
`команда`
```

заменяется последовательностью слов, составляющих стандартный вывод команды. Символы перевода строки при этом заменяются пробелами.

Рассмотрим пример. Команда

```
/etc/mount
```

выводит информацию о монтированных устройствах. Каждому устройству в выводе команды соответствует строка из трех слов: имя специального файла (без приставки /dev/), слово 'on', имя

корневого каталога монтированной файловой системы. Построим процедуру dmo:

```
set \ /etc/mount\
for i
do
        /etc/unmount /dev/□i
        shift
        shift
done
```

В отличие от команды /etc/umount, которая демонтирует одно устройство, команда dmo демонтирует несколько устройств. Список монтированных устройств она получает из вывода команды /etc/mount. Этот вывод команда set присваивает позиционным параметрам, а две последовательные команды shift позволяют иметь дело только с именами специальных файлов, которые интересуют нас в данном случае.

Важной особенностью командного языка UNIX является аппарат командной среды. Он позволяет передать вызываемым командам значения выбранных командных переменных. Тем самым увязываются два языка, два уровня программирования: внешний командный язык и язык программирования в системе, например Си.

Командная среда — это, по определению, совокупность пар (имя переменной, значение переменной), передаваемых вызываемым программам. Переменные, включенные в командную среду, иногда называют экспортируемыми. В основном командная среда формируется встроенной командой export. Команда

```
export x1 x2
```

объявляет переменные x1 и x2 экспортируемыми, и они будут передаваться в дальнейшем всем вызываемым программам.

Возможно экспортирование локальное, в пределах одной команды. Его реализует оператор присваивания, помещенный перед именем команды. Например, запись

```
x1=abc cmd arg
```

означает, что команде (программе) cmd, вызываемой с аргументом arg, будет передана через командную среду пара (x1, abc).

Заметим, что переменные, введенные вне процедур или в теле некоторой процедуры, не доступны вызываемым процедурам. Если в вышеприведенном примере cmd — процедура, то текущие командные переменные ей не передаются. Механизм командной среды позволяет обойти это ограничение — экспортируемые переменные передаются вызываемым процедурам.

И наконец, рассмотрим вопрос о целочисленных вычислениях. Командные переменные, как уже говорилось, имеют символьный тип. Если значениями некоторых переменных служат строки цифр, над ними можно производить целочисленные вычисления.

Делается это с помощью спецнальиой команды expr, прнчем используется подстаиовка вывода команды. Запись

```
x=\expr y + z\
```

говорит о том, что значениями перемениых x, y и z служат строки цифр. Комаида expr суммирует y и z и помещает в стаидартный вывод строку цифр, представляющую сумму этих переменных. Затем уже известиый иам механизм подставляет вывод комаиды вместо правой части оператора присваивания.

5. ПЕРЕНОС СИСТЕМЫ

Седьмая версия операциониой системы UNIX, которой посвящеиа киига Р. Готье, стала первой мобильиой версией этой системы. Существовавшая ранее шестая версия UNIX зиачительио слабее седьмой как по своим мобильиым свойствам, так и по ииструмеитальиым возможиостям. В шестой версии миогие программы, совершеиио не зависящие от машииной архитектуры, написаны иа языке Ассемблера. Нет четкого делення системы иа машиимо-завнсимую и машиимо-независимую части, иеоправдаимо часты в программах иа языке Си машиимо-зависимые коиструкции.

Седьмую версию Bell Laboratories объявила мобильиой (portable). В статьях, вошедших в так иазываемую «докумеитацию» по системе, описаиа процедура переиоса UNIX с 16-разрядиой архитектуры ЭВМ PDP-11 иа 32-разрядную архитектуру ЭВМ Interdata 8/32. Опыт переиоса системы иа миогие другие ЭВМ показал, что оиа действительио мобильиа. Некоторые иедостатки седьмой версни, затрудиявшие переиос, были исправлеиы в последующих версиях System 3 и System 5.

Седьмая версия действительио может считаться мобильиой, поскольку измеиеиия, обусловливающие переиос, иесравиимы с полиым перепрограммироваиием системы. Для переиоса UNIX необходимо выполиить следующие действия:

переписать для иовой ЭВМ кодогеиератор компилятора языка Си;

написать ассемблер для новой ЭВМ;

переписать иекоторые фуикции из системиой библиотеки, иаписаиные на языке Ассемблера;

переписать те модули ядра системы, которые либо иаписаны иа языке Ассемблера, либо машиимо-зависимы;

переписать или иаписать заиово автоиомиые программы для загрузки н иачальиого запуска системы;

переписать иекоторые машиимо-зависимые коиструкции в программах, реализующих комаиды системы.

Переиос UNIX иа отечествеииые ЭВМ и построеиие операциоииой системы, совместимой с UNIX, требует решеиия ряда проблем. Некоторые из них связаиы с особеиностями структурной

организации системы, другие — со свойствами новых технических средств, русификацией представления символьной информации и т. п.

5.1. ОГРАНИЧЕННОСТЬ ВИРТУАЛЬНОГО АДРЕСНОГО ПРОСТРАНСТВА

Проблема, о которой пойдет речь, возникает при реализации системы на 16-разрядных ЭВМ, каковыми являются все модели СМ ЭВМ отечественного производства, кроме СМ-1800 и ее модификаций. Длина виртуального адреса в таких машинах совпадает с основной длиной целого числа и составляет 16 разрядов. Размер виртуального адресного пространства не может превышать 2^{16} бит, т. е. 64 килобайтов. Это тот максимальный диапазон адресов, в котором должны работать как ядро системы, так и каждый из процессов. В этот диапазон должны укладываться все сегменты программы: процедурный, сегмент данных (вместе с динамическим сегментом) и стек.

Канонические версии UNIX, предназначенные для работы на моделях семейства PDP-11, рассчитаны в основном на модели 44, 45 и 70. Особенность таких моделей — наличие двух адресных пространств — инструкций и данных, каждое по 64 килобайта. Программы для этих моделей строятся так, что машинные инструкции (в случае UNIX — процедурный сегмент) располагаются в пространстве инструкций, а данные (в случае UNIX — остальные сегменты) — в пространстве данных. Как ядро, так и процессы UNIX могут оперировать на этих моделях с данными объемом до 64 килобайтов и выполнять программы, достигающие такого же объема.

В СМ ЭВМ аналога этим моделям PDP-11 нет. Единственной отечественной ЭВМ, обладающей указанными свойствами, является «Электроника-79», совместимая с PDP-11/70. В остальных случаях приходится иметь дело с одним виртуальным пространством и учитывать общее ограничение на размер любой программы, равное 64 килобайтам.

В принципе UNIX работает на всех моделях PDP-11 с диспетчером памяти, а не только на трех вышеуказанных. Ограничение на диапазон виртуальных адресов проявляется двояко: не работают некоторые обслуживающие программы и нельзя построить ядро системы для большой конфигурации технических средств.

Большинство обслуживающих программ системы умещается в 64 килобайта. Однако есть несколько программ, требующих раздельных пространств инструкций и данных. Важнейшие из них — компилятор Фортрана 77, переносимый компилятор языка Си и верификатор программ lint.

Сжатие этих программ для машин с одним виртуальным пространством может происходить двумя путями. Во-первых, можно уменьшать размер программы, уменьшая ее функциональную нагрузку, например, исключать какие-то конструкции входного языка компилятора Фортрана 77, двигаясь назад в сторону Фортра-

на IV. В результате получится компилятор, воспринимающий язык, занимающий промежуточное положение между этими версиями Фортрана. Такой путь, конечно, мало привлекателен.

Второй путь — введение в UNIX оверлейных структур и внесение таких структур в большие программы, что связано с переделкой редактора связей ld и в дальнейшем потребует от программиста продумывания оверлейной структуры при написании программы и явного ее указания при компоновке модулей. Технически такой путь достаточно ясен и, как показывает опыт, требует немного времени.

Однако так же поступить с ядром системы не удастся — здесь оверлейный механизм нужно имитировать. Фактически нужно вынести из адресного пространства ядра какие-то компоненты и отображать их виртуальными адресами только в те, очень короткие промежутки времени, когда эти компоненты находятся в работе.

Выносить из адресного пространства ядра можно как программы, так и данные. Если выносятся программы, то ими, как правило, являются драйверы, которые в этом случае называются загружаемыми. Примером может служить операционная система RSX-11 фирмы DEC. Выносить драйверы, а не другие программные компоненты ядра удобнее потому, что интерфейс драйверов с остальной частью ядра достаточно формально специфицирован и точек соприкосновения между ними сравнительно мало.

Что касается UNIX, то предпочтительнее выносить из ядра не драйверы или какие-либо другие программы, а структуры данных. Прежде всего это относится к программно реализованной кэш-памяти, служащей системным буфером для блокориентированных дисков и лент. Как показывает опыт, вынос из адресного пространства ядра кэш-памяти не очень сложная процедура, причем позволяет убить сразу двух зайцев. С одной стороны, достигается искомая цель: уменьшается размер ядра системы. С другой стороны, когда кэш вынесена, нет необходимости отображать виртуальными адресами сразу всю кэш-память.

Отображение, как уже говорилось, устанавливается на короткий промежуток времени, например, только на время переписи блока из кэш в память процесса или обратно. Здесь достаточно иметь обращение только к переписываемому блоку. Поскольку кэш больше не в ядре и не отображается полностью виртуальными адресами, общий размер кэш лимитируется уже не размером виртуального пространства, а размером физической оперативной памяти. Тем самым вынесенная из ядра кэш может достигать существенно большего размера по сравнению с исходной ситуацией (когда она была в ядре), что повышает производительность системы.

Аналогичные рассуждения справедливы и по отношению к другим структурам данных, например, индексным дескрипторам, которые целесообразно выносить из адресного пространства ядра.

Как и во многих других системах, представление символьной информации в UNIX основано на ASCII. В этом 7-битном коде имеются прописные и строчные буквы латинского алфавита. При создании совместимой с UNIX системы для отечественных ЭВМ в символьный код нужно внести русские буквы, сохранив латинские. Существует так называемая совмещенная таблица кодов, полученная путем совмещения наборов H0 и H1 стандартного кода КОИ-7. Она совпадает с кодом ASCII с той лишь разницей, что позиции строчных латинских букв используются для кодирования прописных русских букв. Код совмещенной таблицы, так же как и код ASCII, 7-битный.

Поскольку в латинском алфавите 26 букв, а в русском — 31 (без ъ и ё), в совмещенной таблице по сравнению с ASCII отсутствуют еще пять символов ({} / ~ ^). В операционных системах СМ ЭВМ (РАФОС, ОС РВ, ДОС КП и др.), а также в языках программирования, применяемых в них, эти символы не встречаются, так что их отсутствие практически не заметно. (Уровень развития терминалов, печатающих устройств и средств обработки текстов был еще недостаточно высок и отсутствие строчных букв не считалось криминалом.)

Сейчас положение меняется. Появляются внешние устройства, рассчитанные на прописные и строчные буквы обоих (латинского и русского) алфавитов. Совершенствуются средства обработки текстов, расширяются возможности подготовки и обработки на ЭВМ произвольной текстовой информации. Сказанное с полным правом можно отнести к UNIX, так как средства форматирования текстов этой системы широко известны. С помощью UNIX пишутся статьи и книги, изготавливается печатная продукция.

При переносе UNIX на отечественные ЭВМ целесообразно сохранить все достоинства средств обработки текстов и приспособить их для работы с русским алфавитом. Но отсутствие строчных букв здесь уже недопустимо. Отметим также, что 5 вышеперечисленных символов, не вошедших в совмещенную таблицу, широко используются в командном языке системы и языке Си; прежде всего это относится к фигурным скобкам и вертикальной черте.

Таким образом, совмещенной таблицы кодов недостаточно для построения операционной системы, совместимой с UNIX и допускающей наличие русских букв. Нужен код, с прописными и строчными буквами обоих (латинского и русского) алфавитов.

Код ASCII — подмножество каждого из двух стандартных кодов КОИ-7 (полный) и КОИ-8. Полный код КОИ-7 состоит из двух наборов: H0, содержащего латинские буквы и совпадающего с ASCII, и H1, содержащего русские буквы. Поскольку код 7-битный, в каждый момент времени возможна работа только с одним набором. Переход от H0 к H1 выполняется с помощью кодов «выход» и «вход». Первоначально текущим (рабочим) набором служит набор H0.

Если предположить, что в системе могут обрабатываться только однородные тексты (т. е. содержащие либо латинские, либо русские буквы), то переход от ASCII к КОИ-7 может быть осуществлен достаточно легко. Однако на практике довольно часто встречаются смешанные тексты, в частности системная документация.

Поскольку в КОИ-7 русские и латинские буквы представлены одинаковыми кодами (к какому алфавиту относится данный код, определяет ближайший к нему предшествующий символом «вход» или «выход»), традиционные программы UNIX с КОИ-7 могут восприниматься неоднозначно. Например, команда

```
ed — c file
```

интерпретирует содержимое файла file в виде последовательности символов. Эту команду можно дать в форме

```
od — c file+offset
```

причем файл будет читаться не с начала, а со смещения offset. Так как od не анализирует текст файла для выявления в нем символов «вход» и «выход» и реализует смещение offset с помощью системного вызова lseek (прямой доступ к файлу), она будет интерпретировать все буквы как латинские. Даже если научить ее распознавать символы «вход» и «выход», неопределенность все равно останется: к какому алфавиту принадлежат буквы, встречающиеся от смещения offset до ближайшего символа «вход» или «выход», неизвестно. Устраивать же поиск предыдущего символа «вход» или «выход» — все равно, что заменить прямой доступ последовательным, что неприемлемо.

Другой пример неопределенности — контекстный поиск в редакторе ed. В целом можно сказать, что распознавание символов «вход» и «выход» во внутреннем представлении текста — задача настолько утомительная, что кандидатуру КОИ-7 приходится отклонить.

Остается достаточно разумный выход — ввести 8-битный код для внутреннего представления символьной информации в системе. Этим кодом может быть стандартный код КОИ-8, только желательно упорядочить русские буквы согласно русскому алфавиту. Код ASCII остается подмножеством такого кода, так что совместимость с UNIX (по кодам) сохраняется. Заметим, что речь идет о внутреннем представлении символов; если терминал или печатающее устройство работают в другом коде (например, полном КОИ-7), то соответствующая перекодировка выполняется драйвером устройства. В частности, если нужен выход в линию связи, допускающую только 7-битные коды, то драйвер выполняет перекодировку на обоих концах линии.

Переделка всех программ системы UNIX на 8-битный код — это довольно кропотливая работа. Среди всего того, что требуется сделать для переноса UNIX на отечественные ЭВМ, это самый сложный этап, сравнимый с разработкой ядра системы для новой ЭВМ.

М. И. Беляков

ЛИТЕРАТУРА

1. Беляков М. И., Ливеровский А. Ю., Наумов Б. Н. и др. Инструментальная мобильная операционная система ИНМОС, — Прикладная информатика. Вып. 2, М., 1984, с. 51—68.
2. Fielder D. The Unix Tutorial, Byte, August 1983, с. 186—219, September 1983, p. 257—278, October 1983, p. 132—156.
3. Oakley B. W. Tomorrow's computers (Great Britain). IEEE Spectrum, 1983, 20, № 11, p. 69—72.
4. Organic E. I. The Multics System: An examination of its Structure The MIT Press, Cambridge, Mass, 1972.

ОГЛАВЛЕНИЕ

Ричард Готье

РУКОВОДСТВО ПО ОПЕРАЦИОННОЙ СИСТЕМЕ UNIX

Книга одобрена на заседанин секцни редсовета по электронной обработке данных в экономике 29 августа 1983 г.

Зав. редакцней А. В. Павлюков. Редактор О. Б. Степаиченко
Мл. редактор О. А. Ермнлина
Техи. редакторы: К. К. Буквлова, И. В. Завгородняя
Корректоры: Т. М. Колпакова, Г. А. Башвриина, Л. Г. Захарко
Худож. редактор Ю. И. Артюхов. Переплет художнпка А. Н. Жданова

НБ № 1597

Сдано в набор 9.01.85. Подписано в печать 14.06.85. Формат 60×90¹/₁₆. Бум. кн.-журн. Гарнитура «Литературная». Печать высокая. Усл. п. л. 14,5. Усл. кр.-отт. 14,5. Уч.-изд. л. 14,5. Тираж 30 000 экз. Заказ 1477. Цена 1 р. 30 к.

Издательство «Финансы и статистика», 101000, Москва, ул. Чернышевского, 7.

Областная типография управления издательств, полиграфин и книжной торговли Ивановского облисполкома, 153628, г. Иваново, ул. Типографская, 6.